LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SÓLO PROGRAMADORES

Revista especializada para usuarios de PC

AÑO 3. Nº 17
1250 PTAS.
ARGENTINA 9'50 $
CHILE 3000 $



Y además:
- DEMOS: Efecto lente
- FORMATO FLI/FLC
- UNIX: Sockets
- GRANDES SISTEMAS: DB2
- TÉCNICA HASH

Hipertexto: Inclusión de sonido

Sonido: Ficheros S3M

RAY TRACING: LA REFRACCIÓN

Linux: Instalaciones a medida

Creación de un compilador: Analizadores Léxicos

Contiene DISQUETE con fuentes y rutinas

¡ÚLTIMO PLAZO!

TORNEO DE PROGRAMACIÓN DE DEMOS
250.000 Ptas. en efectivo para el ganador

LENGUAJE HTML: DOCUMENTOS MÁS PROFESIONALES

TOWER COMMUNICATIONS, S.R.L.




DESDE SIEMPRE TODOS HEMOS DESEADO
MIRAR A TRAVÉS DE WINDOWS



CARTA DEL DIRECTOR



ENERO 1996. Número 17
SÓLO PROGRAMADORES es una publicación de Tower Communications

Editor
Antonio M. Ferrer Abelló
Director Editorial
Mario de Luis García
Director de Producción
Carlos Peropadre
Directora Comercial
Carmina Ferrer

Director
Mario de Luis García
Redactor Jefe
Carlos Doral Pérez
Coordinador técnico
Eduardo Toribio Pazos
Publicidad
Mª Eugenia González
Colaboradores
Fernando de la Villa, Fernando J. Echevarrieta, Pedro Antón. Juan M. Martín, Luis Martín, José María Peco, José Carlos Remiro, Enrique de Alarcón, Agustín Guillén, Juan Ramón Lehmann, Héctor Martínez, Álvaro Silgado, David Aparicio, Ignacio Cea, Enrique Castañón, Daniel Navarro,Francisco Otero.
Maquetación
Fernando García Santamaría
Tratamiento de imagen
María Arce Giménez
Servicio Técnico
Oscar Rodriguez
Servicios Informáticos
Digital Dreams Multimedia, S.L.
Ilustraciones
Miguel Alcón
Secretaría de Redacción
Rosa Arroyo
Suscripciones
Erika de la Riva
Redacción, Publicidad y Administración
C/ Marqués de Portugalete, 10
28027 MADRID
Telf.: 741 26 62 / Fax: 320 60 72
Filmación
Filma Dos S.L.
Impresión
G.D.B.
Distribución
SGEL

Depósito legal: M-26827-1994
ISSN: 1134-4792

LOS PROGRAMADORES QUE VIENEN

Es un hecho que las cosas están cambiando muy deprisa. Como decía un amigo mío "antes un programador era alguien capaz de escribir unas cuantas líneas de código seguidas que hicieran algo..." Ahora es diferente. Los programadores autodidactas van dejando paso a las nuevas generaciones de universitarios con carreras técnicas que son valoradas cada vez más en los procesos de selección de las empresas.

Estructuración, técnicas de programación, algoritmos, TODO está ya escrito en buenos libros y sólo hay que buscar en el lugar adecuado. La formación en muchos casos puede llegar a equivaler a gran parte de la experiencia, aunque en la práctica no sea así.

Los nuevos programadores van a tener poco que ver con los pioneros de la informática. Cada vez son más las herramientas de autor que permiten construir programas de apariencia vistosa y funcionamiento *admisible*. Si bien es cierto que una aplicación en Icon Author, o Macromind Director no puede competir con su réplica en C++, (subrutinas en ensamblador incluidas), los cierto es que cada vez son más los desarrolladores de productos (especialmente los llamados multimedia), los que han sustituido los métodos tradicionales por otros en los que el rendimiento de la aplicación empeora considerablemente en favor de la reducción en el periodo de desarrollo. Por contra, los programadores *de siempre* rehuyen este tipo de herramientas por considerar que son la base de una nueva generación de *desarrolladores light*. Y así de paradójicas están las cosas, los que *vienen*, cada vez saben más de todo en general pero muy poco de lo *particular*. Cabría preguntarse si las titulaciones actuales, la de programador incluida, están formando en realidad una nueva generación de analistas, incapaces de resolver problemas concretos en los que haya que ponerse manos a la obra. Tal vez por ello las empresas sigan valorando como el 90 % del curriculum la experiencia adquirida en trabajos anteriores, porque consideren que la formación académica, lejos de ser la adecuada para un programador, sirve tan solo para poner las bases teóricas de lo que se aprenderá con el trabajo de cada día.

Cambiando de tema, este pasado mes de diciembre se ha celebrado en Dinamarca The Party, la gran fiesta de los demomaniacos para Amiga y Pc. En el momento de escribir estas líneas, todavía no ha llegado el reportaje que SP ha preparado sobre el evento, pero será asignatura para el próximo número.

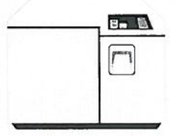
Sobre nuestro concurso de demos, anunciar que el fallo final se tomará en 10 de febrero, y los resultados, (demos incluidas) se publicarán en el número de marzo. La fecha límite e improrrogable para la recepción de trabajos (y esta vez si que no podemos alargarlo más) será el 30 de enero. Desde aquí animamos a los indecisos a unirse a nuestra party de SP.

Nada más por este mes, nos vemos el próximo número.

MARIO DE LUIS

SUMARIO

NÚMERO 17



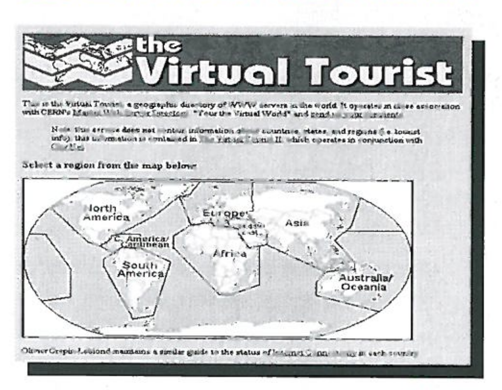


DESTACADO



Nueva entrega del curso de Ray Tracing, uno de los cursos con más aceptación de la revista, este mes se estudia la refracción, gracias a la cual será posible diseñar imágenes más reales y complejas

EL LENGUAJE HTML



En el número pasado se inició el estudio en profundidad del lenguaje HTML de hipertexto, con éste el lector será capaz de crear sus propias páginas Web

Micro Focus Challenger 2000

Micro Focus ha presentado sus novedades tecnológicas en las área cliente/servidor, orientación a objetos, descarga del mainframe y problemática del año 2000 en la Primera Conferencia de Usuarios de la Península -España y Portugal-, celebrada en Sitges (Barcelona) durante los días 7 y 8 de noviembre.

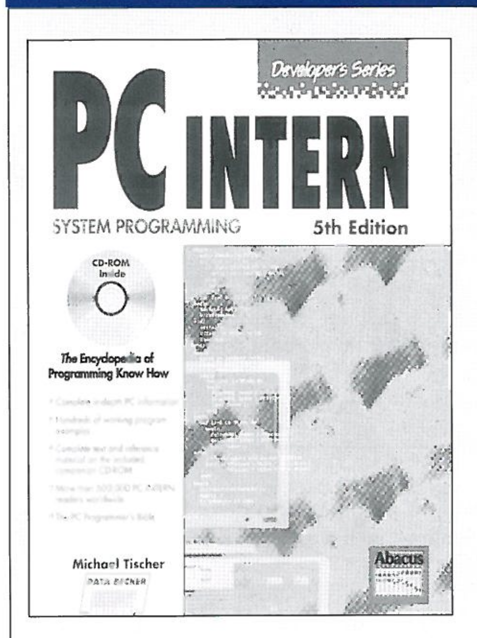
Entre las novedades presentadas destaca Micro Focus Challenge 2.000, una solución desarrollada por la empresa para evitar la pérdida de datos y aplicaciones en el cambio del milenio a consecuencia del cambio de dígitos a "00". Según un estudio proporcionado por Micro Focus y realizado por la consultora Gartner Group, la corrección de errores generados por este problema absorberá inversiones mundiales por valor de 12,6 billones de pesetas, hasta el año 2.000. Micro Focus Challenge 2.000 es un software completo que abarca desde la detección de las aplicaciones y sistemas que sufrirán este problema hasta la implementación y prueba de la solución al mismo.

Otra de las novedades es la última versión de Object Cobol, basado en la tecnología de orientación a objetos que incluye OLE2. Object Cobol permite la incrustación de aplicaciones como Microsoft Word o Excel, permitiendo un desarrollo rápido aprovechando el código existente.

En el área de cliente/servidor Micro Focus permite el desarrollo de aplicaciones para PC con interfaces gráficos de usuario, y proporciona una total transparencia en las comunicaciones y el acceso a bases de datos remotas.

Para más información:
ITEM Comunicación
Angela Herce
Tel: (977) 22 26 10

LIBROS



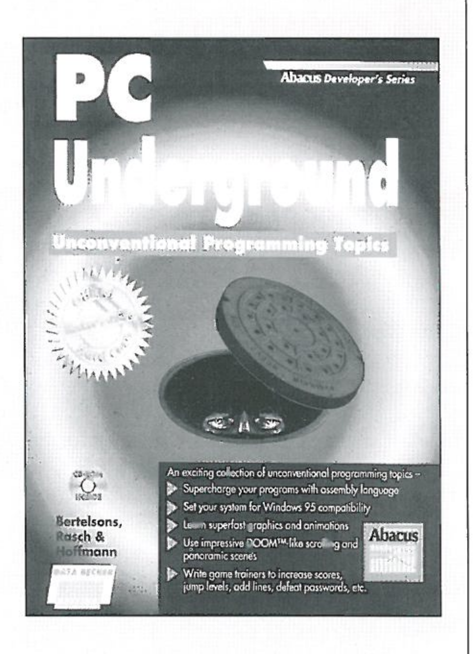
PC Intem. System programming

PC Intern es la quinta edición en inglés del conocido libro de Michael Tischer. Se trata de una completa guía que descubre a los programadores los aspectos más recónditos de la programación de sistemas. En sus páginas se puede encontrar todo lo referente a la programación del hardware de los PCs, como la programación del controlador de interrupciones (PIC), del acceso directo a memoria (DMA), el teclado, el ratón y el CD-ROM. Otros importantes temas tratados son la utilización del modo protegido, el NetBIOS y las estructuras y funciones del DOS. Viene acompañado por un CD-ROM con el texto completo del libro y todos los programas de ejemplo.

Editorial: Abacus
Autor: Michael Tischer
739 páginas
Idioma: Inglés
Precio: 9725 Ptas.

PC Underground Unconventional Programming Topics

Como su propio título indica, este libro cubre los aspectos menos convencionales de la programación en el PC. La obra está dirigida a programadores cuyos intereses se centran en la programación de juegos, demos y otras aplicaciones semejantes. Se tratan temas tan dispares como la optimización de código en ensamblador, los formatos gráficos GIF y PCX, protección contra copia y programación de las tarjetas Sound Blaster y Gravis UltraSound. El libro incluye multitud de código en Pascal y en ensamblador con atractivos programas de ejemplo. Incluye un CD-ROM con todos los programas .



Editorial: Abacus
Autor: Bertelsons, Rasch y Hoffmann
547 páginas
Idioma: Inglés
Precio: 4.005 Ptas.



EL ELEFANTE MULTIMEDIA

Leí de pequeñito un cuento hindú, en el que a un sabio brahmán se le preguntaba sobre la verdadera naturaleza de la sabiduría. El brahmán convocó a un grupo de ciegos y les colocó junto a un elefante. "Describid lo que es para vosotros un elefante", dijo. Uno de ellos palpó la pata del paquidermo y sentenció: "Un elefante es una alta columna". Otro se encontraba bajo la barriga del animal, y replicó: "No, hombre. Un elefante es un gran techo blando y convexo". A un tercero le había caído en suerte la trompa, y terció: "No tenéis ni idea, tíos. Un elefante es una gran serpiente de sangre caliente". El brahmán concluyó entonces que la verdadera sabiduría consiste en ver un todo donde los demás no perciben más que fragmentos.

Me da en la nariz que la tecnología multimedia se encuentra en este momento en su estadio elefántico. A la explotación de esta nueva veta ha llegado un tropel de compañías y profesionales de procedencia muy diversa, y cada uno habla de la feria según le va en ella. Los que vienen del mundo editiorial, por ejemplo, tienden a pensar en los productos multimedia como "libros electrónicos", y a juzgar por lo que están lanzando algunas de las editoriales, deben de creer que para producir un buen producto multimedia no hay más que tomar un buen título de la estantería y someterlo a un cambio de soporte. El resultado: una especie de libro redondo con un agujero en el medio, muy caro, con ilustraciones de mala calidad, letra difícil de distinguir, y que exige para su lectura un aparato que cuesta doscientas mil pesetas. Aquéllos que pertenecen por tradición al mundo de los proyectos audiovisuales, lo prefieren plantear como un "audiovisual interactivo", que es una especie de documental en vídeo que alguien ha fragmentado previamente en cien cintas VHS de un minuto para que el usuario cargue cada una de ellas en su vídeo y pulse "Play" según su mayor o menor interés, lo cual se cepilla todos los aspectos positivos que podría tener un cierto grado de sana pasividad en el espectador: la capacidad de cambiar por un ratito el centro de gravedad del cuerpo, apoyar las dorsales en el respaldo del asiento y dejar, receptiva y agradablemente, que le cuenten a uno una historia bien contada. Y así todo.

Pero ésta es una revista de programadores. Para la mayoría de los programadores, un producto multimedia es un programa. Un programa fácil de escribir, con muchos más datos que código. Y los programadores, que tenemos muchas noches en vela a nuestras espaldas, tendemos a pensar que un buen producto es un buen programa, con rutinas de gráficos rápidas y algoritmos de búsqueda efectivos, y bueno, los datos no son más que eso, datos, son la paja que se añade a los programas para que el disco llene sus preceptivos seiscientos cincuenta megas, no vaya a ser que el comprador descubra que más de la mitad de la irisada superficie de su cedé está lisa como un espejo perfecto, y sienta que le han tomado el pelo. Y cuando el comprador prueba su cedé, efectivamente se siente estafado, pero no porque el disco no esté a rebosar, sino porque los datos, sencillamente, no interesan a nadie.

No descubriremos ningún secreto: esto de la publicación multimedia no es más ni menos que un nuevo soporte, una página casi en blanco que está esperando su nuevo lenguaje. Una forma nueva de decir cosas, que exige, como todas, que primero se tenga algo que decir. Una agrupación de medios que permite, al que sepa hacerlo, combinar algunas virtudes de varios medios tradicionales, y siempre pagando por ello el peaje de un mercado precoz y saturado, con estándares efímeros e inseguros y compradores que, en su mayoría, son ya gatos escaldados, y no se tragan más el cuento que hasta ahora se ha ido oyendo: "No deje que su familia pierda el tren del futuro: si no se entera de qué va nuestra nueva y revolucionaria tecnología interactiva virtual, la gente le va a señalar con el dedo por la calle y dirá, Mira, ése es Martínez, el pelagatos al que no le importa mandar a sus hijos de cabeza al foso de los cocodrilos del fracaso escolar y las drogas, porque aún no se ha gastado doce mil pesetas de nada en el último cedé que deleita a la par que instruye". Si es que algún día funcionó ese chantaje de la tecnología por la tecnología, hace ya tiempo que ha dejado de asustar al personal.

Conque, si nos queremos meter en este tinglado de la multimedia, a arremangarse la camisa, escupirse en las manos y pensar qué tipo de producto podemos ofrecer con nuestros medios. Atención al público objetivo: habrá que escoger uno y ponerse a su servicio, porque el mercado multimedia está madurando y diversificándose, ya no es esa sopa primigenia de chavalines tecnochiflados y padres tecnoparanoicos del pasado inmediato. Atención a los contenidos: habrá que elegirlos de calidad, actualizados, distribuidos con equilibrio y una estructura clara. Atención a los valores de la interactividad: deberán guiar al usuario de sillón con progresividad y talento narrativo, y deberán dar al usuario inquieto e investigador la libertad que necesita para escoger el camino y descubrir nuevas relaciones. Atención al diseño: debe ser efectivo, mantener la atención, conseguir que el usuario encuentre lo que busca y reforzar la imagen de calidad. Atención al precio: aunque todos los cedés tengan el mismo grosor y diámetro, lo que contienen no tiene por qué costar lo mismo. No debemos pedir por cada producto más de lo que estemos dando, la gente no es tonta y nos posicionaría como vendedores de alfombras. Y sobre todo: nuestro producto debe ser un producto de calidad porque tenga un contenido de calidad y esté expuesto con calidad. No porque sea multimedia e interactivo y contenga más de quinientos megas de información. Eso ya no impresiona a nadie.

Cesar Astudillo

Bay Networks anuncia una estrategia integrada RMON para sus productos

La compañía Bay Networks ha anunciado la disponibilidad de una nueva suite de herramientas de diseño y análisis RMON para la monitorización de capa de red a nivel de la empresa. Mediante los agentes Advanced Analyzer y el software de administración de red Optivity Design & Analysis de Bay Networks los clientes pueden utilizar ahora las matrices de tráfico de 3 capas, un componente clave del emergente estándar RMON2.

Bay Networks también ha anunciado la disponibilidad de los routers de acceso remoto Access Node Hub (ANH) y Access Node (AN) con RMON incorporada.

RMON proporciona un mecanismo para recopilar las estadísticas de dispositivos y de la capa MAC, lo que permite a los administradores de red ver el flujo de tráfico dentro de un segmento de la red, pero no proporciona una vista del tráfico cuando pasa los límites del router. El estándar emergente RMON2 incrementa el nivel de visibilidad, permitiendo a los administradores de red ver el flujo del tráfico en la red de la empresa desde la red hasta las capas de aplicaciones. Los administradores de red pueden visualizar el tráfico cuando circula a través de la empresa en cada capa del modelo OSI, facilitando enormemente la administración de los entornos cliente/servidor.

Optivity Design and Analysis 6.0 ya está disponible, mientras que la funcionalidad RMON está disponible como opción para los routers Access Node Hub y Access Node.

Para más información:
Bay Networks
Ricardo Miranda-Naón
Tel: (91) 742 50 13

Intel presenta el Pentium más rápido para la informática móvil

Intel presenta el Pentium más rápido para la informática móvil

Intel Corporation ha presentado un procesador Pentium a 120 MHz que ofrece a la informática móvil prestaciones similares a las de las máquinas de sobremesa. Arroja un índice iCOMP de 1.000/120 y es el primer procesador para sistemas móviles fabricado mediante el proceso en geometría de 0,35 micras de Intel. Gracias a este chip de tamaño reducido los ordenadores portátiles podrán proporcionar niveles de prestaciones muy elevados teniendo a la vez un bajo consumo.

El procesador Pentium a 120 MHz funciona a 3,3 voltios empleando componentes corrientes mediante la utilización de la tecnología de reducción de tensión, mientras que su núcleo central funciona a sólo 2,9 voltios. De este modo es capaz de ofrecer el 30% más de prestaciones que un Pentium a 90 MHz, pero con el mismo consumo.

El procesador se presenta a la vez en encapsulado TCP y en encapsulado de cerámica clásico integrando la tecnología de reducción de tensión. Alcanza unas prestaciones SPECint92 de 133 y SPECfp de 99. Tiene una disipación térmica tipo de 2,5 a 3,5 vatios y consume menos de 1 vatio en modo reposo.

Por cantidades de 1.000 unidades cuesta 681 dólares en los Estados Unidos, mientras que el precio del procesador Pentium a 90 MHz es de 341 dólares y el del mismo procesador a 75 MHz es de 204 dólares.

Para más información:
Intel Corporation Iberia
Tel: (91) 308 25 52
Fax: (91) 310 54 60
World Wide Web:
http://www.intel.com/

Acuerdo entre IBM y Bay Networks

IBM Corporation y Bay Networks han anunciado un acuerdo que abarca la promoción conjunta de estándares, las pruebas de interoperabilidad y la comercialización cruzada de los productos Token-Ring de ambas compañías. Por éste acuerdo IBM y Bay Networks trabajarán conjuntamente para proporcionar la conmutación Token-Ring y las soluciones de migración Token-Ring a ATM, incluyendo Token-Ring full duplex y los interfaces estándar para facilitar la aceptación de la industria y del ATM Forum de I-PNNI (Integrated Private Network-to-Network Interface) y MPOA (Multiprotocol Over ATM).

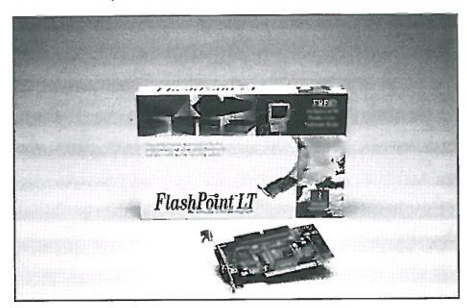
Las dos compañías se han comprometido a asegurar la total interoperatividad entre Switched Virtual Networking de IBM y la arquitectura BaySIS de Bay Networks.

Controlador Ultra SCSI Plug & Play para bus PCI de BusLogic

BusLogic ha introducido en el mercado FlashPoint LT, el primer Host adaptador PCI Ultra SCSI con características SCAM (SCSI auto configurable) nivel 1 y compatible PCI 2.1. Ha sido desarrollado para la última generación de PCs de sobremesa existentes en el mercado y se caracteriza por la alta velocidad de transferencia en 32 bits bus master DMA que transmite a través del bus PCI a una velocidad de hasta 133 megabytes por segundo. Siguiendo las especificaciones SCSI-3 duplica la velocidad de transferencia de datos a través del bus SCSI desde 10 Mb/seg. a 20 Mb/seg. Es fácil de instalar y dispone de GreenLogic, un economizador de energía que reduce el consumo del bus SCSI cuando está inactivo.

FlashPoint LT incluye un paquete de software multimedia de características avanzadas. El FlashPoint Bonus Pack es un conjunto de herramientas de productividad incluido en la versión kit de FlashPoint LT. El Bonus Pack incluye software para CD, McAfee Associates Scan 95H, BusView y Media Rack.

Ya están disponibles los drivers para los principales sistemas operativos del mercado, como Windows 95, Windows NT, DOS y OS/2. FlashPoint LT está disponible a un precio de lista de 28.625 ptas. El kit del adaptador incluye cables internos, drivers, Bonus Pack y la documentación necesaria y está disponible a un precio de 32.375 ptas.



Para más información:
CompuPlacers S.L.
Norma Healy
Tel: (93) 451 55 99
Fax: (93) 454 39 58

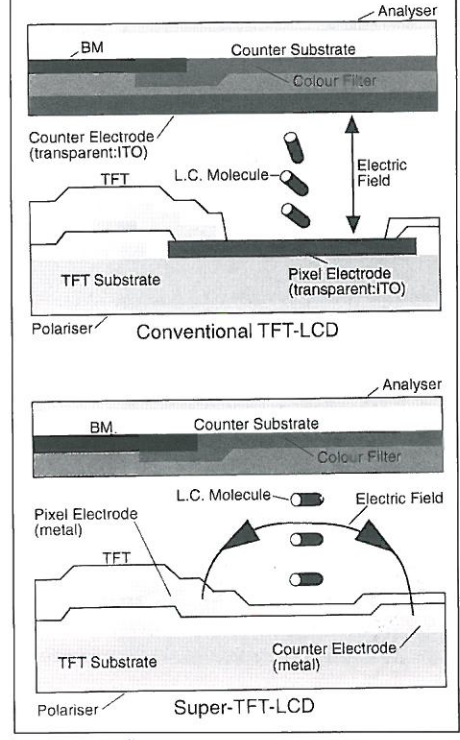
Hitachi anuncia su nueva tecnología de pantallas LCD

Hitachi anunció recientemente su nueva tecnología de pantallas LCD "Super TFT", que introduce importantes mejoras en el ángulo de visualización sobre las pantallas TFT convencionales sin que ello suponga una pérdida de la calidad de imagen. Con esta nueva tecnología las pantallas LCD podrán competir con la tecnología CRT en aplicaciones de múltiple visualización.

Con la tecnología Super-TFT se logra un ángulo eficaz de visualización de 70 grados (desde el normal) en todas las direcciones sin ningún cambio o variación del color. Esto supone una mejora de 3x sobre la tecnología TFT-LCD convencional. Para conseguirlo se utiliza la técnica de conmutación de plano (IPS), en la que las moléculas de cristal líquido conmutan la transmisión de luz al mismo tiempo que mantienen sus ejes longitudinales paralelos al plano de fondo.

Otras aplicaciones en la que es posible encontrar la tecnología Super-TFT son terminales de banco, sistemas de distribuidores, sistemas de punto de venta y otros sistemas de automatización de oficinas.

El primer producto basado en Super-TFT será una pantalla XGA (1024 x 768 pixels) de 13,3" y 262k de color con muestras disponibles en el primer trimestre de 1996 y producción completa a mediados de 1996.



Para más información:
Hitachi Europe
Pedro Aparicio
Tel: (91) 767 27 82 / 92

Nuevo récord de beneficios de Intel

La rápida conversión del mercado al procesador Pentium ha conseguido que Intel alcance una facturación y beneficios récord para el tercer trimestre de 1.995. La facturación de 4.170 millones de dólares para este trimestre que finalizó el 30 de septiembre de 1.995 representa un incremento del 46% en comparación con los 2.860 millones de dólares para el mismo periodo del año 1.994. El tercer trimestre de 1.995 es el dieciseisavo trimestre consecutivo en que los ingresos de Intel aumentan.

El beneficio neto fue de 931 millones de dólares, lo que supone un incremento del 41 por ciento en relación a los 659 millones de dólares del mismo periodo del año anterior y un incremento del 6 por ciento en comparación con los 879 millones de dólares en el segundo trimestre de 1.995. Las ganancias por acción se incrementaron un 38 por ciento con respecto a los 0,76 dólares del tercer trimestre de 1.994 y un 6 por ciento en comparación con los 0,99 dólares del segundo trimestre de 1.995.

Según Andrew S. Grove, presidente y CEO de Intel, en el tercer trimestre de 1.995 se superó un jalón cuando las entregas de Pentium sobrepasaron por primera vez a las del procesador Intel486.

Para más información:
Intel Corporation Iberia
Tel: (91) 308 25 52
Fax: (91) 310 54 60

EL LENGUAJE HTML (II)

Fernando J. Echevarrieta

En el artículo anterior se presentaron las líneas básicas para la realización de documentos HTML sin apartarse de los estándares. Con ello, era ya posible presentar cualquier tipo de información a través del WWW. En el presente, se profundizará más en ciertas particularidades del lenguaje con matices más profesionales, como la realización de *forms* o de imágenes-mapa sensibles.

Abriendo un breve paréntesis con el objeto de estar al día, hay que añadir a la lista de proveedores de hace dos números una nueva entrada: Lander Internet, tlf. 902 363 363 que además, si sólo se desea acceso a WWW lo proporciona de forma gratuita en un número ilimitado de conexiones de media hora de duración. Pero volvamos al tema que nos ocupa.

IMÁGENES MAPA

Uno de los efectos más aparentes y que primero suele apetecer incluir en las propias páginas es el de las imágenes-mapa. Una imagen-mapa consiste en un un gráfico sensible que conduce a uno u otro lugar en función de sobre qué zona del mismo se pulse. De este modo, en servicios como el turista virtual (figura 1), se localizan los servidores WWW por zonas geográficas pulsando sobre los países o regiones correspondientes sobre mapas. Es ya común en la red obtener una u otra información al pulsar sobre distintas cajas en un esquema o, más espectacularmente, acceder a la página personal de cada persona pulsando sobre ella en una fotografía de grupo. Por ejemplo, en http://highland.dit.upm.es la "imagen de la trompeta" es, en realidad, un mapa sensible que conduce a distintos lugares si se pulsa sobre la imagen del autor o sobre otra zona.

El mes anterior, se mencionaba cómo la etiqueta *<IMG>* de HTML admitía un argumento denominado *ISMAP*. Es este argumento el que cualifica una imagen como mapa sensible. Como es obvio, no basta indicar que una imagen es un mapa para que el sistema conozca cuales son las zonas sensibles. Así, será necesario especificar estas zonas en un fichero asociado. Al pulsar el usuario sobre una fotografía cualificada con la etiqueta *ISMAP*, el cliente de WWW enviará las coordenadas al servidor, donde arrancará un programa *CGI* encargado de leer el fichero asociado y procesarlas. Aunque en principio, el usuario puede programar su propio *CGI* para procesar las imágenes, no será necesario ya que todos los servidores incorporan un programa para ello. Por el momento, los más famosos son el programa *htimage* del servidor del *CERN* y el *imagemap* del servidor de *NCSA*. Si va a incluir imágenes-mapa en sus páginas y no ha instalado vd. el servidor de WWW, deberá preguntar al administrador cuál de ellos es el programa de que dispone. Esto se debe a que la forma que tienen de definir las zonas sensibles, si bien muy similar, es suficientemente distinta para hacerlos incompatibles.

Por tanto, para realizar imágenes sensibles deberá seguir los siguientes pasos:

1. Crear un fichero donde especifique las zonas sensibles de la imagen, como se indica en los párrafos siguientes.
2. Especificar *ISMAP* en su llamada a la etiqueta *<IMG>*
3. Convertir la imagen en un enlace que invoque al programa mediante la etiqueta *<HREF>*, como se explicó en el número anterior.

Para ello, deberá indicar como URL el siguiente:

<dir_CGI>/<programa>/<fichero_mapa>

En el presente artículo se profundiza en la realización de documentos HTML más elaborados explicando a fondo la realización de forms, y presentando la forma de realizar imágenes-mapa sensibles e incluir imágenes transparentes y entrelazadas.

donde *dir_CGI* es el directorio en que se encuentran los programas *CGI* de su servidor de WWW (si no lo ha instalado Vd., pregunte a su administrador); *<programa>* será uno de los mencionados *htimage* o *imagemap* (u otro creado por el administrador) y *<fichero_mapa>* será el fichero que ha creado en el primer paso. Así, por ejemplo:

**
<img src=/sp/fotomapa.gif ISMAP>
**

sería el código HTML asociado a la imagen-mapa de la figura 2. Esta página ha sido instalada con el URL: http://highland.dit.upm.es:8000/sp/fotomapa.html para aquellos lectores que deseen comprobar su funcionamiento.

El formato del fichero-mapa es, como se ha mencionado, diferente para cada programa. En el caso de imagemap de NCSA consta de una serie de líneas que definen, cada una de ellas, una figura geométrica con la sintaxis:

figura URL coord-1, coord-2 ... coord-n

donde *figura* es el nombre de la figura geométrica y *coord-i* son las coordenadas de los puntos necesarios para definirla. *URL* determina el URL al que el cliente debe saltar en caso de que las coordenadas del punto sobre el que se ha pulsado caigan dentro del área definida por la figura. Este URL, como en cualquier enlace, podrá ser absoluto o relativo (ver artículo del mes anterior). Así, son posibles las combinaciones:

circle URL cx,cy bx,by

define un círculo donde *cx,cy* es el centro y *bx,by* un punto de la circunferencia límite

poly URL x1,y1 x2,y2 ...

define un polígono donde *xi,yi* son las coordenadas de un máximo de 100 vértices

rect URL x1,y1 x2,y2

define un rectángulo donde *x1,y1* son las coordenadas de la esquina superior izquierda y *x2,y2* son las de la esquina inferior derecha

point URL x,y

define un punto de coordenadas *x,y*. No es recomendable la utilización de esta figura.

Por último, existe una figura denominada *default* a cuyo URL asociado se saltará si el punto sobre el que se ha pulsado no pertenece a ninguna de las áreas englobadas por las figuras definidas.

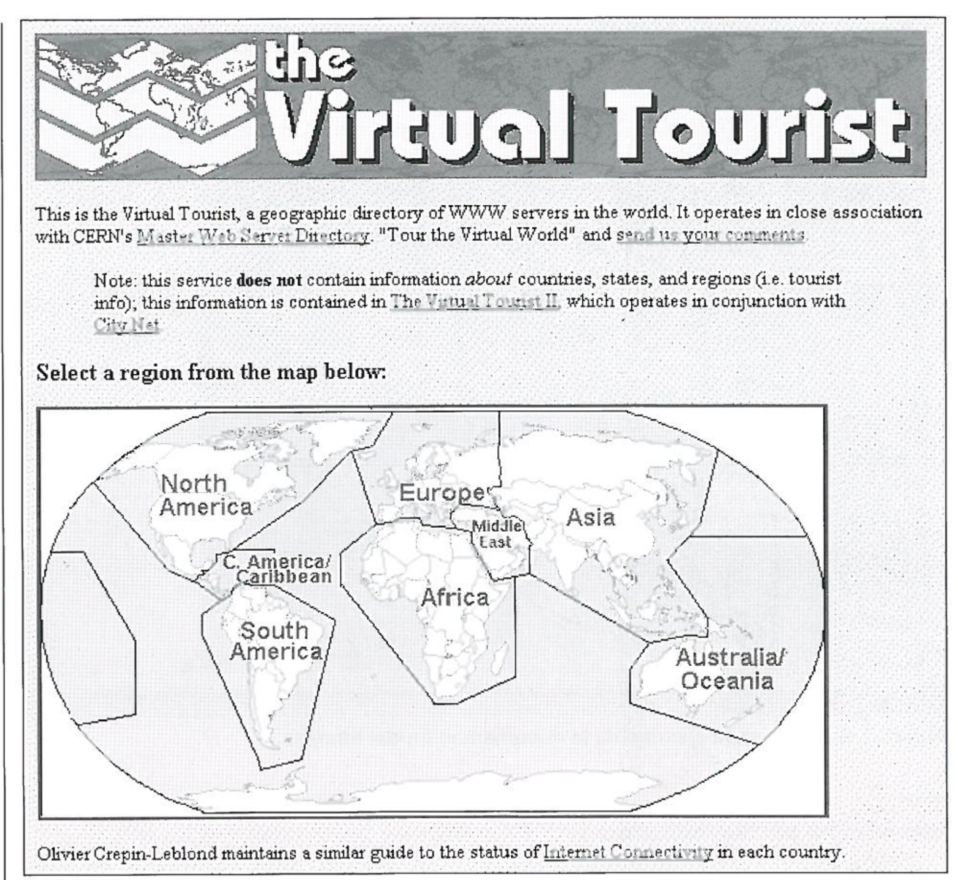


Figura 1. Mediante la imagen-mapa del turista virtual se accede a los distintos países pulsando sobre ellos

Un ejemplo de fichero-mapa de *imagemap* para la página html de la figura 2 podría ser:

default
http://highland.dit.upm.es:8000/sp/fotomapa.html
rect
http://hoohoo.ncsa.uiuc.edu/docs/tutorials/imagemapping.html 38,46 173,150

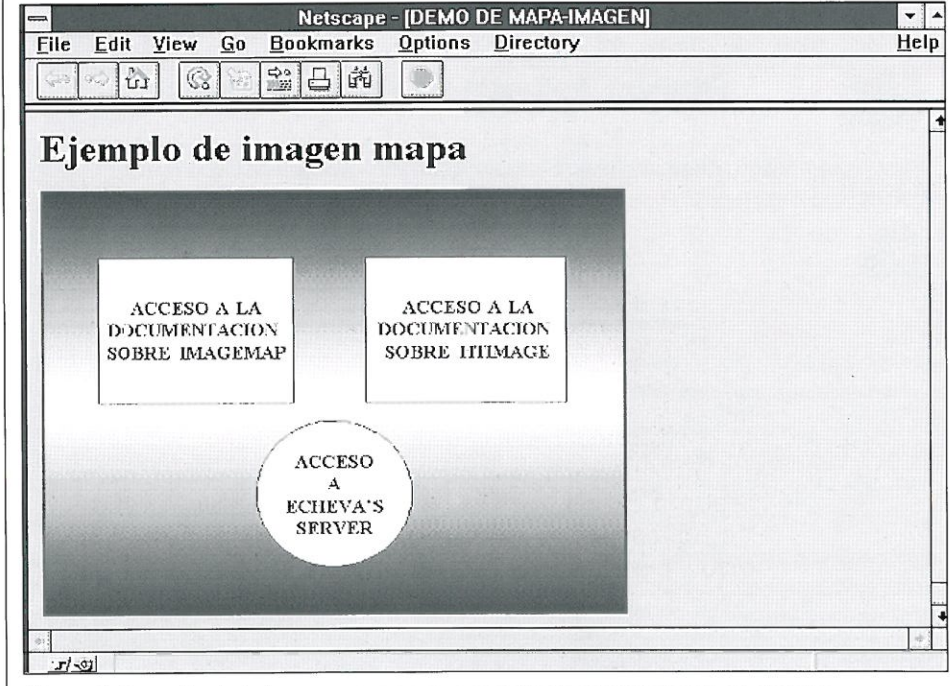


Figura 2. Aspecto de la imagen-mapa correspondiente a los listados de ejemplo.

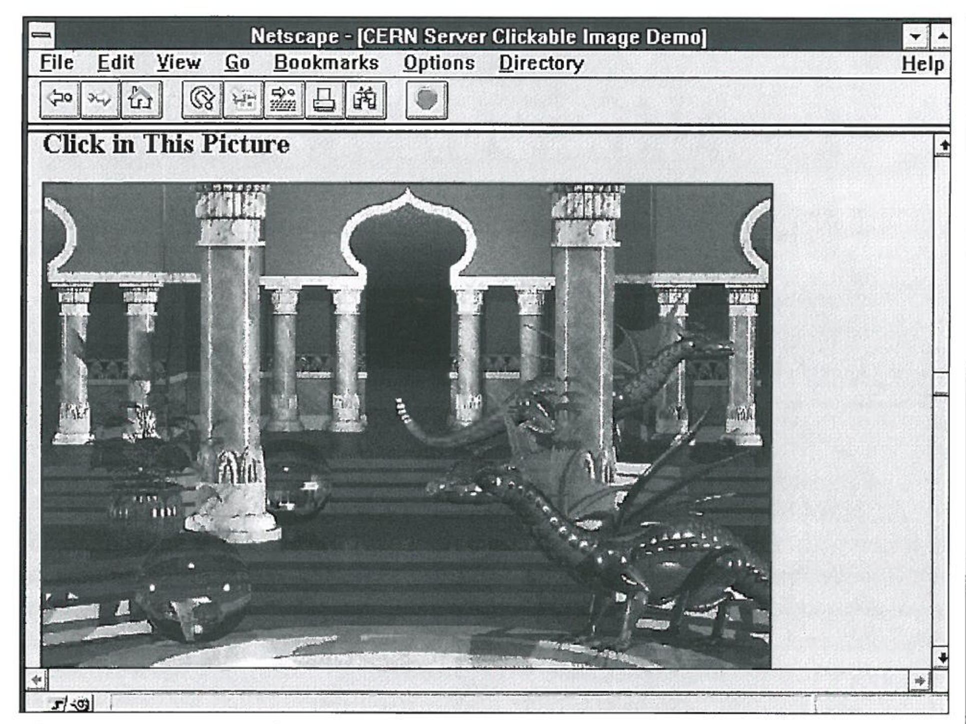


Figura 3. Imagen-mapa ejemplo de la documentación del htimage.

rect
http://www.w3.org/hypertext/WWW/ Daemon/User/CGI/HTI mageDoc.html
223,47 361,150
circle http://highland.dit.upm.es:8000
200,214 252,214

El programa *htimage*, por su parte, lee un fichero análogo en el que se definen figuras similares. Además de permutar el orden de los datos, no hay que olvidar que enmarca las coordenadas entre paréntesis. En esta ocasión, le sintaxis es: figura coordenadas URL

La sintaxis para especificar en este caso las figuras es:
circle (x,y) r URL
donde *(x,y)* son las coordenadas del centro del círculo y r su radio
rectangle *(x1,y1) (x2,y2) URL*
esta vez las coordenadas corresponden a dos esquinas cualesquiera con la condición de que sean opuestas

Netscape - [Spain Internet Resources]
File Edit View Go Bookmarks Options Directory Help
Spain Internet Resources
[About this map | Register new resources | Search Spain e-mail addresses | Spain Info | UJI Home Page | New resources]
Click only inside the blue rectangles.

Figura 4. Un ejemplo de la combinación de transparencia e imagen-mapa.

polygon *(x1,y1) (x2,y2) ... (xn,yn) URL*
donde las parejas *(xi,yi)* definen vértices adyacentes

Por supuesto, también existe la figura *default*, obligatoria. Estas figuras pueden ser abreviadas mediante *circ, rect, poly* y *def* respectivamente.

El fichero-mapa equivalente al anterior para la figura 2 para el programa *htimage* sería:
default
http://highland.dit.upm.es:8000/sp/ fotomapa.html
rectangle (38,46) (173,150) http:// hoohoo.ncsa.uiuc.edu/docs/tutorials/ imagemapping.html
rectangle (223,47) (361,150) http:// www.w3.org/hypertext/WWW/ Daemon/User/CGI/HTImageDoc.html
circle (200,214) 52
http:// highland.dit.upm.es:8000

En caso de que dos o más de las áreas definidas en el fichero mapa se solapen, se tomará como válida la primera que figure en el fichero. Por ello, si en la imagen en cuestión existen distintas figuras en diferentes planos ocultándose unas a otras, es una buena práctica definir los contornos completos de las mismas pero comenzando por las que están en primer plano. Esto también se puede utilizar para simplificar las zonas geométricas de un mapa, por poner un ejemplo, dejando que los contornos más complejos vayan quedando definidos por las áreas que los rodean.

En la figura 3 se puede ver la imagen demo de la documentación del programa *htimage* disponible en línea, cuyo fichero-mapa se ha incluido en el disco de la revista con el nombre dragons.con.

Como ayuda a la realización de estos ficheros-mapa, se han creado ya algunas utilidades shareware, como el programa hotspots para windows.

Por último, es necesario mencionar que han surgido nuevas propuestas como las de Spyglass para la realización de imágenes-mapa que no impliquen la existencia de un programa CGI en el servidor, al trasladar la complejidad al cliente. De este modo, las coordenadas de las zonas sensibles figuran en el mismo código de la página HTML y es el cliente el encargado de procesar las pulsaciones. Si desea más información al

respecto podrá encontrarla en http://www.spyglass.com/techspec/img_maps.html.

IMÁGENES ENTRELAZADAS Y TRANSPARENTES

Una técnica bastante interesante en la presentación de páginas WWW es la utilización de imágenes transparentes y entrelazadas. Ambas características se encuentran soportadas por la última versión del estándar GIF, la GIF89a. Sin embargo, muchos de los programas que trabajan con GIFs únicamente soportan la versión anterior, GIF87a. Un programa shareware muy completo capaz de trabajar con GIFs entrelazados es el Paint Shop Pro (PSP).

Una imagen que se crea como entrelazada aparece en un browser que soporte este modo con una resolución pobre que va mejorando según va llegando el resto de la imagen. En browsers que no acepten esta posibilidad no presentarán ningún problema debido a que, como cualquier otra imagen se mostrará una vez haya sido recibida completamente.

Uno de los efectos más espectaculares en las páginas de web, son las "imágenes flotantes" que parecen formar parte de las mismas. Estas imágenes se integran perfectamente con los fondos sea cual sea el color o dibujo de estos, ya que el color de fondo de la imagen es interpretado como transparente.

En la figura 4 se puede observar una imagen transparente que a la vez es un mapa sensible realizada con gran sentido estético. Ésta es otra de las facilidades del estándar GIF89a. Sin embargo, encontrar programas que la gestionen es tarea más difícil. Uno de ellos es el programa giftool de David Koblas, que se proporciona con fuentes. Otro de los programas más utilizados por los creadores de páginas WWW es el giftrans, que puede obtenerse por ftp anónimo de ftp.rz.uni-karlsruhe.de en /pub/net/www/tools/giftrans.c. Estos programas toman como entrada una imagen GIF y como parámetro el color que se desea se convierta en transparente y devuelven la versión 89a correspondiente.

LOS FORMS: A FONDO

Una de las carencias de la primera versión de HTML era la imposibilidad de que el navegante enviara información a los servidores. Su utilidad, evidente: desde el envío de comentarios, a la cumplimentación de formularios para solicitud de información, telecompra, realización de encuestas pasando por la posibilidad de interactuar con programas introduciendo datos que incluso podrían utilizarse a través de pasarelas con la totalidad de los servicios de información que ofrece Internet.

Así pues, una de las primeras propuestas fue la inclusión del mecanismo hoy conocido como forms. La interfaz de forms de HTML permite la realización de formularios que el usuario puede cumplimentar para enviar al servidor, donde será procesada por un programa que cumpla el estándar CGI, que en la mayoría de las ocasiones generará dinámicamente páginas HTML que serán enviadas como respuesta al cliente.

En el presente artículo, se estudiará la parte del cliente, dedicada a la realización de páginas, dejando el tema de la programación CGI para los próximos artículos. En cualquier caso, el lector, podrá comprobar desde el primer momento el fruto de su trabajo ya que al ser un form una página HTML, será posible visualizarlo con cualquier browser, con todas sus características, como ventanas de selección con scroll, o botones pulsables sin necesidad de que realmente exista el programa encargado de procesar su información. Únicamente será necesaria su presencia cuando el form defina un botón de envío o submit y éste sea pulsado. Sólo en este momento, y no antes, el cliente enviará toda la información recogida por el form en el estado de sus botones y en sus ventanas y el servidor WWW activará el CGI indicado para procesarla.

Para la realización de este tipo especial de páginas interactivas, HTML define las siguientes etiquetas: *<form>...</form>*, *<input>*, *<option>*, *<select>...</select>*, y *<textarea>...</textarea>*, careciendo las cuatro últimas de significado si se emplean fuera de la zona englobada por la primera.

TABLA: FORMS
<form [ACTION=][METHOD=]>...</form>
<input [type=text|password|checkbox|radio|submit|reset] [name=][value=][checked=][size=] [maxlength=]>
<select [name=][size=][multiple]>

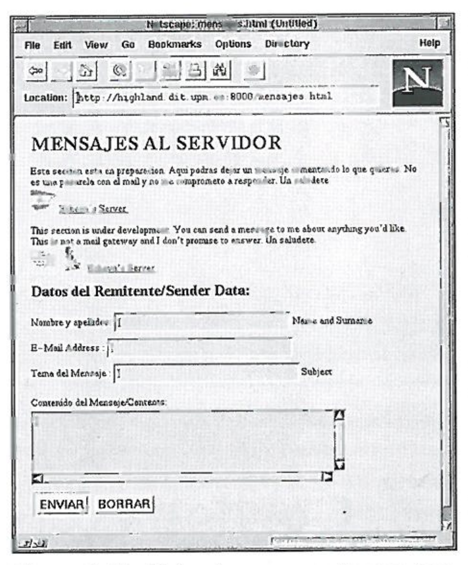


Figura 6. Un típico form con entradas text y textarea.

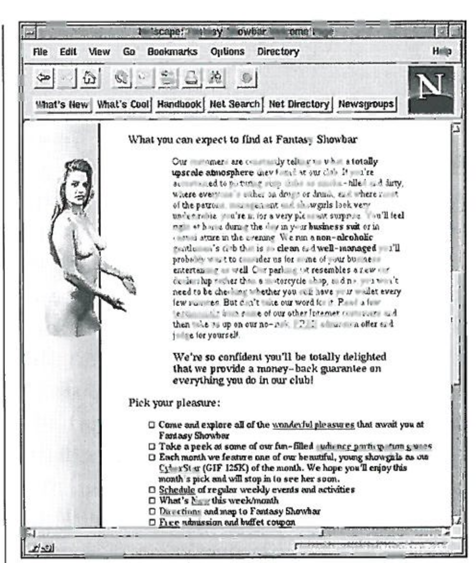


Figura 5.

Si las zonas geométricas se solapan, se tomará la primera que aparezca en el fichero

<option [selected]>
<textarea name=""
[rows=][cols=]></textarea>

La primera etiqueta, doble, se utiliza para definir el comienzo y fin de un formulario. En un documento HTML puede haber varios, pero no está permitido encadenarlos ni solaparlos. La etiqueta <form> admite dos opciones. La necesidad de una de ellas se ve de forma intuitiva: una vez recibida la información habrá que hacer algo con ella. Así, la opción *action*, indica el programa CGI encargado de procesar esta información. La segunda opción, menos intuitiva, indica el modo en que se debe realizar la transmisión de información a este programa. Su nombre es *method* y puede tomar dos valores, *post* o *get*. Más adelante en la serie se profundizará en estos mecanismos. Por el momento, se mencionará que lo más recomendable será siempre utilizar la opción *post*. Como ejemplo de declaración de un form se podría tener:

<form action=/cgi-bin/CGImessages
method=post>
...
</form>

CAMPOS Y ÁREAS DE TEXTO

Dentro de cada form, la información se introduce a través de campos definidos mediante la etiqueta *<input>* (obsérvese que no existe etiqueta de cierre). A través de la misma, se definen variables pertenecientes a unos tipos predefinidos y a las que se puede asignar un valor por defecto.

El tipo de una variable se define mediante la opción *type* que puede adoptar los siguientes valores:

text: indicando variable de texto.

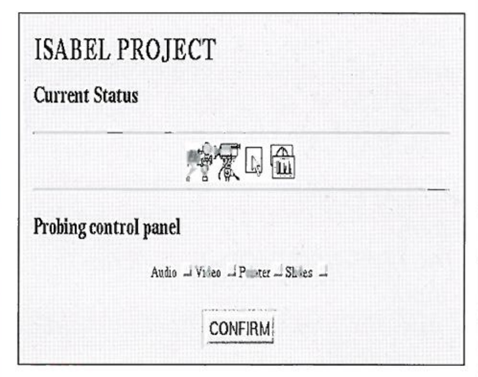


Figura 7. Otro ejemplo con el uso de entradas checkbox, en este caso para el control remoto de una aplicación.

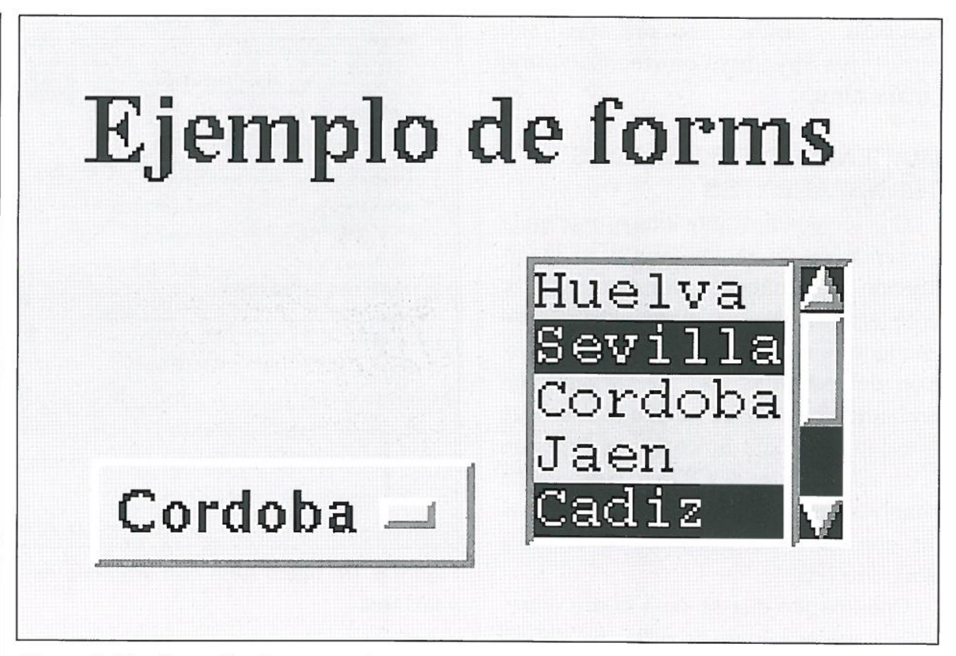


Figura 8. Menús realizado con <select>.

password: idéntica a *text* salvo en que el que eco que produce al escribir es de asteriscos y no aparece cuando se consulta al browser la URL a solicitar.

checkbox: variable binaria que puede estar seleccionada o no radio: permite definir varios campos con un sólo nombre pero seleccionar sólo uno.

submit: genera un botón de acción que envia el formulario al ser pulsado.

reset: genera otro botón de acción que al ser pulsado restaura todos los campos a sus valores por defecto o los deja en blanco si no los tienen.

En caso de que una misma página contenga varios forms, cada uno deberá contener su botón submit, por lo que únicamente se podrá procesar uno de ellos.

Existe otro tipo de campo denominado hidden que permite almacenar variables de estado, que no se mostrarán al usuario. Es importante aclarar que un campo hidden no se presenta en pantalla, pero no se trata de un campo secreto ya que todos los browsers pueden visualizar la fuente HTML de la página.

Este campo es especialmente útil debido a que como se mencionó en el primer artículo de la serie, HTTP es un protocolo de transmisión sin estado, por lo que si una aplicación debe conservar una memoria de estado, ésta debe ser dinámica y circular constantemente, ya que no puede ser almacenada. Así, por ejemplo, si es necesario presentar varios forms al mismo usuario, se podría preguntar su nombre en el primero y encargar al programa CGI que lo procesara que generara el siguiente form dinámicamente incluyendo el nombre de usuario en una variable *hidden*. Pero sobre ésto se profundizará en su momento, si no comprende aún el significado de este tipo de variable no se preocupe, en los próximos artículos se le presentará como algo obvio.

Es posible asignar valores por defecto a los campos de un form

El nombre de la variable se asigna mediante la opción *name*. Así, para definir un botón que pueda pulsarse para indicar, por ejemplo que se desea más información, se podría escribir:

<input type="checkbox" name=
"masinfo" value="on">

que equivaldría a la definición de una variable booleana de nombre *masinfo* asociada a la pulsación o no del botón. No hay que olvidar que cuando programemos CGI, será un programa

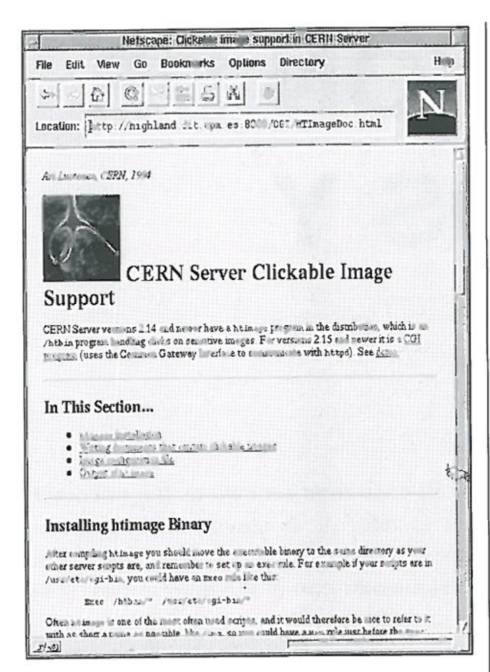


Figura 9. Página de documentación de htimage del CERN.

en el servidor el que reciba la información de que existe esta variable masinfo con valor "on", o sin valor.

El valor por defecto se especifica a través de la opción *value*, de distinto significado según el tipo de la variable. Así, para *text* o *password*, indica un texto por defecto; para *checkbox* o radio, el valor que se le dará a la variable si se pulsa el botón; y para *submit* y *reset*, indica la leyenda que aparecerá sobre los correspondientes botones cuando se le presenten al usuario.

Existen otras posibles opciones para la etiqueta *input*. Así, *checked* indicará que un botón *checkbox* o radio debe aparecer pulsado por defecto (el usuario, si lo desea, tendrá que "des-pulsarlo"); *size*, que indica el ancho con que debe aparecer un campo *text* o *passwd* y *maxlength*, que define el número máximo de caracteres que estos admitirán. Ejemplos válidos serían:

<input type="password" name="passwd" size=8 maxlenght=8>
<input type="checkbox" name="tieneCD" value="puesclaro" checked>

Una de las formas más utilizadas para enviar información a través de forms, son las áreas de texto. Mediante la etiqueta *<textarea>...(texto por defecto)...</textarea>*, es posible definir una ventana-recuadro de texto con barras de scroll horizontales y verticales donde el usuario podrá escribir lo que desee; casí siempre, un mensaje. El tamaño de este recuadro podrá ser especificado mediante las opciones *rows* y *cols* (filas y columnas) y, al igual que con un campo *input*, será necesario darle un nombre mediante la opción name.

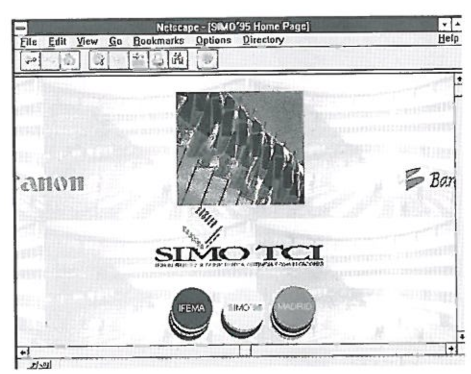


Figura 10. En esta página del SIMO, se puede apreciar cómo los logotipos de las empresas son imágenes transparentes.

VENTANAS DE SELECCION

En ocasiones, se presenta al navegante la posibilidad de elegir entre varias opciones y si éstas no son pocas, la utilización de *checkboxes* o *radio buttons* puede llegar a ser tediosa y antiestética. En estos casos, es posible dar un toque de distinción a los formularios mediante la inclusión de ventanas de selección que permitan al cliente navegar por una ventana en la que se le presenten distintas opciones (por ejemplo, para indicar una provincia) permitiéndole seleccionar sólo una en unos casos o varias a la vez en otros. Para ello, HTML incluye la etiqueta *<select>...</select>*. Esta etiqueta puede contar con tres opciones: la ya conocida *name*, de la cual huelga decir nada más;

size: que indica en número de opciones que serán presentadas a la vez, pudiéndose desplazar el usuario por la lista mediante barras de scroll en caso de que el número de opciones sea mayor y

multiple: que no toma valor e indica que es posible seleccionar varias opciones simultáneamente.

La forma de definir cada opción es análoga a la que se empleaba en listas (ver artículo del mes anterior). Donde allí se usaba la etiqueta *<li>*, aquí se empleará *<option>*. Esta etiqueta únicamente admite una opción, *selected*, que no toma valores y es equivalente a *checked* en *<input>*, es decir, presenta la correspondiente opción seleccionada por defecto. Así, por ejemplo:

<select multiple>
<option>Hombres
<option selected>Mujeres
<option>Otros
</select>

presenta la posibilidad de elegir entre hombre y/o mujeres y/o otros habiendo seleccionado ya la opción de mujeres (que será posible "des-seleccionar"). En las figuras adjuntas se pueden ver dos menús tipo <select> para selecciónar provincias andaluzas. El primero de ellos aparece como un botón que al ser pulsado genera un menú *pop-up*, el segundo, con las opciones *MULTIPLE* y *SIZE=5*, muestra una ventana de scroll.

CONCLUSION

El mes próximo, se hará una presentación del manejo de tablas, una de las características más cambiantes y en evolución desde su aparición en el primer *draft* de HTML3.0.

Para ello, se explicará la implementación de Netscape comentando las "posibles" diferencias con lo que "podría" ser el documento "definitivo". También se abandonarán algo los estándares explicando las extensiones de Netscape en materia de control de presentación, fondos y tipos de letra, y actualización dinámica de documentos. En definitiva, todo aquello que dará un toque de distinción a nuestros documentos.

Con ello, se cerrará el ciclo, dedicado al lenguaje HTML y la serie caminará hacia la instalación de servidores y la programación CGI.

BIBLIOGRAFÍA

- Web Sights, IEEE SPECTRUM, Sep.1995, pág. 87. Richard, Comerford
- NCSA imagemap tutorial http://hoohoo.ncsa.uiuc.edu/docs/tutorials/imagemapping.html
- Clickable image support in W3C httpd http://www.w3.org/hypertext/WWW/Daemon/User/CGI/HTImageDoc.html
- Spyglass Client Side Imagemaps http://www.spyglass.com/techspec/img_maps.html
- HTML Revisited, SUN EXPERT, May. 1995, pág. 28. Peter Collinson
- Spinning the Web: How to Provide Information on the Internet. Andrew Ford. Ed. International Thomson Publishing
- Giftrans http://melmac.harrisated.com/transparent_images.html

INDEXACIÓN DE ARCHIVOS Y HASHING

José C. Remiro

Los archivos con organización secuencial y directa son en muchas ocasiones los únicos disponibles en los lenguajes de programación de alto nivel. Estos dos tipos de organización no son muy adecuados, por sí solos, para resolver el problema de recuperar la información contenida en un determinado registro conocido el valor de uno o varios campos.

Por ejemplo, si se intenta crear un archivo secuencial que contenga el isbn, el título, el autor y el tema de los libros que pertenecen a una biblioteca, para saber si se dispone de una título en particular habría que inspeccionar uno a uno los registros de este archivo, en media, se tendría que recorrer la mitad del archivo hasta encontrarlo y en el peor de los casos la totalidad del archivo para decidir que no se encuentra registrado. Si se almacena esta información en un archivo de acceso directo, el resultado sería el mismo a menos que se pudiera hacer corresponder de alguna forma la posición ocupada por el registro dado el título que se desea buscar.

Los archivos con organización indexada y la técnica del hashing, que serán tratados en este artículo, facilitarán la localización de un registro dado un conjunto de valores pertenecientes a determinados campos que se denominarán campos claves o simplemente clave.

Por regla general los lenguajes de programación de propósito general no disponen de archivos indexados, en el presente artículo se describen varias formas de poder implementarlos.

HASHING

Este método también se conoce como búsqueda por transformación de claves. Consiste en transformar una clave dada, tanto si es numérica como si es alfanumérica, en un número que será la posición ocupada por el registro dentro de un archivo de acceso directo.

El caso mas sencillo es hacer coincidir el valor de la clave con la posición que ocupa el registro en el archivo. Así, y siguiendo con el ejemplo inicial, si se numeran los libros disponibles en la biblioteca y la clave fuese dicho número podríamos localizar el libro mediante el número de registro correspondiente. En la mayoría de las situaciones estas condiciones ideales no se dan pues o bien la clave es alfanumérica o el rango de valores de la clave, aunque numérico, no coincide con el rango de las posiciones de los registros en el archivo.

De esta forma, el primer problema a resolver es encontrar una función que dado un valor de la clave lo transforme obteniéndose la posición que debe ocupar el registro con dicha clave dentro del archivo de acceso directo.

Si se supone que la clave asociada al archivo para almacenar libros es el campo título y éste es de tipo cadena con una longitud de 20 caracteres, los posibles valores de este campo son las variaciones con repetición de 26 caracteres posibles (caracteres alfabéticos más el carácter blanco) tomadas de 20 en 20, es decir, 26^{20}. Si se dispone de 1000 libros, habrá que crear una función que dado un título proporcione un valor entre 0 y 999. Puesto que el número de posibles títulos, supera al número de registros que habrá que almacenar es evidente que la función de transformación de claves produzca idénticas posiciones para libros con distintos títulos, a esta anomalía se la denomina colisión de claves.

La primera cuestión a solucionar es seleccionar una función que dada una estructura de clave transforme el valor de ésta en un número adecuado de registro. Por otro lado, es deseable que

la función evite en lo posible la colisión de claves, distribuyendo de una forma homogénea los registros a lo largo del archivo.

Existen diferentes métodos para crear una función de transformación de claves. Evidentemente, si la clave es alfanumérica se hará corresponder a cada posible carácter de la clave un número. Los métodos mas habituales aplicados sobre el número que se ha hecho corresponder a la clave para transformarlo en la posición que ocupará el registro son los siguientes: Truncamiento, plegamiento, aplicación del módulo y el método del cuadrado.

El método de truncamiento consiste en seleccionar unas determinadas posiciones de la clave, construyendo a partir de éstos un número de registro. Así, por ejemplo, si se ha hecho corresponder el número 2837925 a una clave determinada y disponemos de un archivo de acceso directo con 1000 registros, podemos seleccionar los dígitos de posiciones primera, cuarta y sexta obteniendo de esta forma el número de registro 272, que se encuentra dentro del rango de números de registros admisibles. Este método es muy rápido y sencillo, pero desgraciadamente hay un gran número de colisiones, por ejemplo, los números de clave 2587028 y 2007029 colisionarían con el anterior.

El método de plegamiento consiste en agrupar posiciones del número de la clave y mediante operaciones aritméticas sobre estos agrupamientos obtener el número de registro. Siguiendo con el anterior ejemplo y agrupando los dígitos de dos en dos podríamos sumarlos para obtener la posición . Ahora la posición del registro que se hará corresponder a la clave sería: 02 + 83 + 79 + 25 = 189.

El método del módulo consiste en tomar el resto que resulta de dividir el número de la clave entre el número de posiciones de que dispone el archivo de acceso directo, obteniéndose así un número entre 0 y el número de registros del archivo menos 1. Por ejemplo, si tomamos el número de clave 2837925, la posición que se le hace corresponder si el archivo de acceso directo dispusiera de 1000 registros sería de 2837925 mod 1000 = 925. Es aconsejable si se utiliza este método utilizar como divisor un número primo cercano al tamaño del archivo pues se obtiene una distribución mas uniforme de las claves.

Por último, el método del cuadrado consiste en calcular el número asignado a la clave y truncar los dígitos a derecha e izquierda de manera conveniente. Por ejemplo, el cuadrado de 2837925 es 8053818305625, tomando los dígitos de posición seis, siete y ocho obtendríamos el número de registro 183.

```
funcion transforma_clave (clave: cadena(20)): entero
 aux: entero
 acumulador: entero
 base: entero
 i: entero
 exponente: entero
 primo: entero
inicio
 primo = 997
 const = 0
 base = 10
 exponente= 0
 i = 0
 aux = 0
 mientras i < longitud(clave)
    i = i + 1
    aux = ASCII(clave(i)) - ASCII (' ')
    cont = cont + 1
    Si cont = 5
      entonces
        acumulador = acumulador + aux * base ^exponente
        exponente = exponente + 1
        aux = 0
        cont = 0
    fin Si
  fin mientras
  Si cont # 0
    entonces
      acumulador = acumulador + aux * base ^exponente
  fin Si
 transforma_clave = acumulador mod primo
 fin
```

Cuadro 1.

Evidentemente los métodos se pueden combinar para obtener la función de transformación de claves, aunque es mejor disponer de una función lo mas sencilla posible.

Una vez elegida la función de transformación de clave habrá que solucionar el problema de las colisiones. Para este fin, se incluirá un campo de estado dentro del registro del archivo, este campo se utilizará solamente para gestionar las operaciones sobre los registros siendo irrelevante su contenido para el usuario. El campo de estado solamente podrá tomar los tres siguientes valores: Ocupado, Libre o Borrado. El proceso de las diferentes operaciones sobre el archivo se describen a continuación.

• Para insertar un registro, se transformará su clave obteniéndose la posición donde deberá situarse. Si el registro de dicha posición tiene el valor en el campo de estado libre o borrado el registro se emplazará allí, marcando el campo de estado como ocupado. Si por el contrario, el registro de la posición calculada tiene el campo de estado con valor ocupado se inspeccionarán los registros de forma secuencial con dirección hacia el final del archivo, el primer registro que se encuentre con el campo de estado libre o borrado será la posición donde se inserte el registro, una vez situados los datos en dicho registro se marcará como ocupado. Es interesante observar como una colisión tiene como efecto que registros que no tendrían por que colisionar lo harán por el traslado de registros a otras posiciones que a priori no les correspondían.

```
procedimiento crear_archivo (var arch: archivo libro)

 i, num_reg: entero
 reg: libro

inicio
 num_reg = 999
 Asociar (f, 'Archivo')
Abrir (f, 's')
 estado = 'L'
 desde i = 0 hasta num_reg
        Escribir (f, reg)
Cerrar(f)
fin
```

Cuadro 2.

• Para buscar un determinado registro a través de su clave, se transforma ésta para obtener la posición que le correspondería en el archivo, si la clave de búsqueda no coincide con la del registro que actualmente se encuentra almacenado en dicha posición (en este caso cuando se insertó el registro se produjo una colisión), se deberá inspeccionar secuencialmente y hacia el final del archivo hasta que o bien se encuentra o bien se llega a un registro con el campo de estado con valor libre.

```
procedimiento buscar ( var arch: Archivo libro; clave: cadena(20);
                        var posicion: entero; var encontrado: logico)
 aux, primo: entero
 fin_busqueda: logico
 reg_libro: libro
inicio
 aux = transforma_clave (clave)
 Apuntar (Arch, aux)
 encontrado = falso
 Mientras (no encontrado ) y (no fin_busqueda)
    Leer (Arch, reg_libro)
     Si (clave = reg_libro.titulo) y  (reg_libro.estado = 'O')
        entonces
          encontrado = verdadero
        en otro caso
          Si reg_libro.estado = 'L'
             entonces
               fin_busqueda = verdadero
             en otro caso
               Si reg_libro.estado = 'B'
                  entonces
                     aux = (aux + 1) mod longitud (Arch)
                      Apuntar(Arch, aux)
               fin Si
          fin Si
     fin Si
 fin Mientras
 posicion = aux
fin
```

Cuadro 3.

• Para eliminar un registro conocida la clave, ésta se transforma para conocer el lugar que debería ocupar, se busca según el proceso descrito con anterioridad y una vez encontrado, se marca a través del campo de estado como borrado. Obsérvese que si no se dispusiera de la marca de borrado se perdería el rastro de aquellos registros que colisionaron con el que actualmente se desea borrar y que fueron desplazados hacia posiciones consecutivas.

Para los anteriores procesos, se tendrá en cuenta que el siguiente registro al último del archivo será el primer registro del archivo, por esta razón es conveniente que al menos un registro del archivo se encuentre siempre libre pues podría darse el caso de estar el archivo con todas sus posiciones ocupadas y por tanto el proceso de búsqueda no tendría fin.

En el cuadro 1 se muestra un ejemplo de función de transformación de claves, combinando varios de los métodos expuestos anteriormente. Supuesto que la clave es una cadena de hasta 20 caracteres, lo que hace esta función es agrupar los caracteres consecutivos de 5 en 5, calculando para cada uno de ellos la diferencia entre su código ASCII y el código ASCII correspondiente al carácter blanco (se supone que los únicos caracteres permitidos en la clave son los caracteres alfabéticos en mayúsculas y el carácter blanco), una vez sumados estos códigos se les multiplica por una potencia de 10 dependiendo de la posición que ocupen, así al primer grupo de 5 se les multiplica por 10^0, al segundo grupo por 10^1 y así sucesivamente hasta el cuarto grupo que es multiplicado por 10^3. Una vez obtenido el número, se calcula el resto de dividirlo entre el número 997, el número primo inferior más cercano a 1000 que será el tamaño del archivo a utilizar.

El cuadro 2 muestra el procedimiento para crear el archivo de 1000 registros que contendrá la información. Aparte de los campos relevantes para el usuario que utilizará la información, se ha incluido el campo estado que contendrá 3 posibles valores: 'L' (libre), 'O' (ocupado) y 'B' (borrado), que se utilizará como se describió con anterioridad. Cuando se crea el archivo los campos que contienen la información para el usuario será indefinido, marcándose el estado de todos los registros como 'L' (libre).

Por último, el cuadro 3 muestra un procedimiento para buscar en el archivo, dada la clave del registro correspondiente, devolviendo en el parámetro posición el número de registro donde debería encontrarse y en el parámetro lógico encontrado si éste se encuentra realmente o no. Es importante comprender el bloque de instrucciones de la instrucción mientras, pues en éste se trata la búsqueda si se ha producido una colisión de claves.

Los procedimientos para eliminar y grabar un determinado registro se dejan al lector, pues disponiendo del procedimiento buscar son bastante fáciles de implementar, pues para borrar basta con localizar la posición ocupada por el registro y modificar el campo estado a 'B' y para escribir si no se encuentra un registro de idéntica clave (en cuyo caso el procedimiento buscar devuelve en el parámetro encontrado el valor falso y en el parámetro posición la posición del pri-

mer registro que se encuentre libre o borrado), basta con transferir la información a grabar en el registro de la posición devuelta por el procedimiento buscar y asignar el valor 'O' (ocupado) en el campo estado.

INDEXACIÓN

Al igual que el método de transformación de claves, la indexación de un archivo permite acceder a la información de un registro conocido el valor de la clave.

La implementación mas general de este método consiste en almacenar en un archivo de acceso directo los registros que contienen la totalidad de la información, a este archivo se le denominará de almacenamiento principal. La disposición de los registros dentro de este archivo es, en principio, indiferente. Para auxiliar en la búsqueda de un determinado registro se dispone de un archivo, también de acceso directo, que contiene básicamente el valor de todas y cada una de las claves de los registros que se encuentran en el archivo de almacenamiento principal, junto con la posición que ocupa el registro. A este archivo se le denomina archivo de índices.

La idea básica para la localización de un registro es muy sencilla. Se inspecciona el archivo de índices en busca de la entrada que coincida con el valor de la clave del registro buscado. Una vez localizado, se obtendrá la posición donde se encuentra situado el registro dentro del área de almacenamiento principal.

La localización de un determinado registro se ve relentizada por la búsqueda de la clave en el archivo de índices, por tanto si se desea que el acceso sea lo suficientemente rápido, será necesario mantener ordenado el archivo de índices con el fin de poder aplicar un algoritmo de búsqueda sobre este archivo ordenado.

Además, si el archivo de índices es bastante grande, el proceso de búsqueda se relentizará puesto que el traslado de información desde el dispositivo de almacenamiento secundario a memoria central es muy lento en comparación con el acceso de la CPU a la memoria central. En la implementación desarrollada en este artículo se dispondrá de un array que hará de índice de primer nivel. En él se dispondrá de los pares (valor_clave, posición_indice) donde la posición_índice se corresponderá con el número de registro del archivo de índices donde se encuentra dicho valor de la clave. Evidentemente, el archivo de índices crece de forma dinámica pues lo habitual es dar de alta y de baja registros, sin embargo el array tiene un número fijo de posiciones, la solución es distribuir solamente valores de claves que disten un valor determinado previamente calculado (por ejemplo, dividiendo el número de registros entre el número de posiciones de que dispone el array). Así, por ejemplo si el archivo de índices contiene 1000 registros y disponemos de un array de 100 elementos en este se incluirán los valores correspondientes a las posiciones 0, 9, 19 y así sucesivamente.

La búsqueda se realizará en primer lugar sobre el array, buscando el primer elemento que supere a la clave del registro buscado, si esta se encuentra en el elemento j del array, entonces la entrada en el archivo de índices se encontrará entre la posición del archivo de índices indicada por el elemento j-1 del array y la posición del archivo de índices indicada por el elemento j. Por tanto, con este método no se complica en exceso la gestión de índices disminuyendo el tiempo de búsqueda.

Una vez seleccionado el acceso a los diferentes registros, hay que tratar el problema de cómo insertar nuevos registros en el archivo de almacenamiento principal y cómo eliminarlos.

La inserción de nuevos registros no se realizará de forma directa sobre el archivo de almacenamiento principal, pues en la mayoría de los lenguajes de propósito general, cuando se abre un archivo para lectura no se permite añadir nuevos registros al final del archivo, por tanto se dispondrán en un array en el que se almacenarán temporalmente los registros que se den de alta, para al finalizar la ejecución de un programa, incorporarlos al archivo de almacenamiento principal y una vez allí actualizar el archivo de índices, reflejando su incorporación definitiva. Este proceso tiene asociado un problema añadido, si se desea localizar un determinado registro y el valor de una clave no se encuentra en el archivo de índices habrá que realizar una búsqueda secuencial en el array que contiene temporalmente dichos registros. Es evidente, que si se ha producido una serie de altas habrá que hacer una reorganización de los dos archivos, pues al insertar registros sobre el archivo de almacenamiento principal habrá que anotar en el archivo de índices los nuevos registros y posteriormente ordenar el archivo de índices con los nuevos pares introducidos. Además, por tener la estructura array un número no ampliable de elementos, si en un momento dado el número de altas supera el tamaño del array habrá que realizar este proceso aunque haya un usuario utilizando el archivo, el usuario en este caso constatará una disminución en el tiempo de respuesta del programa.

La eliminación de un registro es mucho mas sencilla, pues se puede

los archivos con organización indexada y la técnica del hashing permiten localizar registros de forma eficiente

Estructura de los registros utilizados para la implementación de un archivo indexado

```
/*estructura para el array de índices primario */
par_indice = registro
        clv: cadena (20)
        pos_ind: entero

/*estructura para el fichero de indices */
reg_indice = registro
        clv: cadena (20)
        posicion: entero
        borrado: logico

/*registro para el fichero de almacenamiento principal */
reg_archivo = registro
        isbn: cadena (14)
        titulo: cadena (20)
        autor: cadena (25)
        tema: cadena (15)
```

Cuadro 4.

```
funcion busqueda_aprox (array_indice: par_indice; inf, sup: entero;
                        clave: cadena(20) ): entero
 mitad: entero
inicio
 mitad = ( inf + sup ) / 2
 Mientras ( inf < sup ) y ( Array_indice (mitad).clv # clave )
      Si clave < Array_indice (mitad).clv
         entonces
           sup = mitad - 1
         en otro caso
           inf = mitad + 1
      fin Si
      mitad = ( inf + sup ) / 2
 fin Mientras
 busqueda_aprox = mitad
fin
```

Cuadro 5.

hacer un borrado lógico, introduciendo un campo de estado en el archivo de índices que indique si el correspondiente registro al que apunta está disponible o no. Periódicamente habrá que proceder al mantenimiento del archivo eliminando físicamente los registros correspondientes a los índices cuyo campo de estado sea borrado, eliminando tanto el registro en el archivo de índices como el correspondiente al archivo principal.

En este artículo solamente se presentará el seudocódigo correspondiente a la búsqueda de un registro conocida su clave.

En el cuadro 5 se muestra la función busqueda_aprox, se trata del seudocódigo de una búsqueda binaria aproximada. Esta función toma como parámetros el array que contiene los pares (clv, pos_ind), que indican para una determinada clave donde se encuentra la correspondiente entrada dentro del archivo de índices. Como ya se ha señalado con anterioridad, el array solamente contiene unas cuantas entradas de la totalidad del archivo índice. Para implementar el procedimiento de trasladar las entradas del archivo de índices al array basta con dividir el número de registros del archivo entre el número de elementos del array, trasladando al array los registros cuya posición sean múltiplos de tal cantidad.

```
funcion busqueda_total (clave: caracter (20); Array_indice: array(100) par_indice;
                        inf, sup: entero): entero
 i: entero
inicio
 i = busqueda_aprox (Array_indice, inf, sup, clave)
 Si Array_indice(i).clv > clave
    entonces
      Si i = inf
        entonces
          busqueda_total = - 1
        en otro caso
          busqueda_total = busca_indice (Array_Indice ( i-1 ).pos_ind, clave)
      fin Si
    en otro caso
          busqueda_total = busqueda_indice (Array_indice ( i ).pos_ind, clave)
 fin Si
fin
```

Cuadro 7.

En el cuadro 6 se muestra la función busca_indice, esta función inspecciona el archivo índice secuencialmente a partir de la posición indicada por el parámetro num_reg. Como se muestra en el cuadro 7, la función busqueda_total se encarga de localizar definitivamente el número de registro dentro del archivo de almacenamiento principal, para ello primero llama a la función busqueda_aprox, con lo que obtiene el número de registro del archivo de índices donde debe comenzar la búsqueda, a continuación, una vez conocido dicho valor llama a la función busca_indice para realizar la búsqueda secuencial, de no encontrar el registro con la clave devolverá el valor -1, que no se corresponde con ningún número de registro y que servirá para comprobar si la búsqueda ha tenido éxito o no. Hay que recordar que si se han dado de alta registros, entonces es necesario comprobar que en el array que se habrá dispuesto para tal fin no se encuentra el registro, este array no se puede inspeccionar por un método de búsqueda binaria pues no estará ordenado, siendo necesaria una búsqueda secuencial. Para eliminar un registro, tras buscarlo, basta con asignar el valor verdadero en el campo borrado del registro correspondiente del archivo índice o bien si es un registro que se ha dado de alta y todavía no se encuentra anotado en el archivo índice basta con eliminarlo del array donde se almacenan los nuevos registros dados de alta.

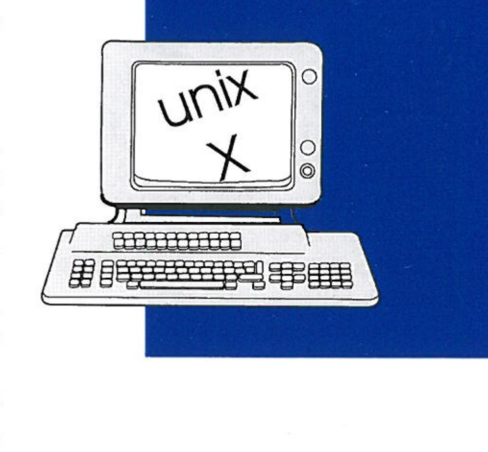
```
funcion busca_indice (num_reg: entero; clave: cadena (20) ): entero
 reg_indice_aux = registro
                    clv: cadena (20)
                    posicion: entero
                    borrado: logico

inicio
 Posicionar (Archivo_Indice, num_reg)
 Leer (Archivo_Indice, reg_indice_aux)
 Mientras (reg_indice_aux.clv < clave ) y ( longitud (Archivo_Indice) > num_reg)
        num_reg = num_reg + 1
        Posicionar (Archivo_Indice, num_reg)
        Leer (Archivo_Indice, reg_indice_aux)
 fin Mientras
 Si (reg_indice_aux.clv = clave) y ( borrado = false )
    entonces
      busca_indice = reg_indice_aux.posicion
    en otro caso
      /*No se ha encontrado el registro con el valor de la clave
        se devuelve un valor ficticio -1 */
      busca_indice = -1
 fin Si
fin
```

Cuadro 6.

ARQUITECTURAS CLIENTE-SERVIDOR

Fernando J. Echevarrieta



Expuesto el mecanismo de control de procesos UNIX así como la API del sistema de ficheros, en el presente artículo se continúa el recorrido por la API del sistema introduciendo una de las interfaces de comunicaciones más empleadas en la programación de comunicaciones.

Si bien en el artículo dedicado a control de procesos se presentaba el mecanismo de Ken Thompson como un método eficaz para la comunicación entre procesos, este método se presenta insuficiente a la hora de comunicar procesos que se ejecutan en distintas máquinas y quizá, en distintas redes. En su momento se explicó que aquel mecanismo se fundamentaba en la utilización de una llamada fork por lo que ambos procesos debían coexistir en la misma máquina simultáneamente y tener un origen común. Por ello, esta técnica se presenta completamente inadecuada a la hora de tratar las arquitecturas cliente-servidor en las que uno y otro programas son totalmente independientes en su origen. Además, ambos procesos podrían encontrarse en máquinas heterogéneas, tanto desde el punto de vista de la arquitectura física como del S.O. Por otra parte, un servidor debería ser capaz de atender a varios clientes y un cliente de solicitar un servicios de varios servidores.

Mediante los named pipes o ficheros FIFO presentados en el número anterior, se solucionaba el problema de la independencia de los procesos. Sin embargo, estos aún debían compartir el sistema de ficheros. Es por ello, que fue necesario crear la abstracción socket como punto de acceso a una comunicación entre procesos independientemente del S.O. ambiente del interlocutor.

A lo largo de este artículo se presentarán las llamadas al sistema necesarias para realización de aplicaciones cliente-servidor basadas en sockets de forma conceptual. Para los detalles de implementación en C se pueden consultar los prototipos de las funciones que se han incluido en la tabla 1. La definición de constantes y estructuras necesarias en C se encuentra en los ficheros de cabecera <sys/socket.h> y <netinet/in.h>. Es conveniente destacar que la abstracción socket se encuentra presente en multitud de S.O. aunque en esta serie se hable únicamente de UNIX. Es el uso de interfaces como ésta, que permiten la comunicación de máquinas heterogéneas lo que permite que un sistema sea calificado como "abierto". El próximo número incluirá como programa ejemplo, un juego del buscaminas en red orientado a conexión que servirá para ilustrar los conceptos expuestos. En posteriores artículos, se presentarán otros medios para crear aplicaciones cliente-servidor, como el mecanismo de RPCs de SUN que, a su vez, se fundamenta en la abstracción socket.

UNIX INTERNETWORKING

Las primeras implementaciones de TCP/IP sobre UNIX actuaban sobre un fichero especial denominado /dev/tcp. A medida que se desarrollaban protocolos más complejos surgió la necesidad de realizar nuevos módulos de biblioteca para definir un interfaz de protocolos. Así surgió la arquitectura cliente-servidor mediante la cual se definen servidores pasivos y clientes activos que crean conexiones hacia los servidores. En cualquier caso, la comunicación entre un cliente y un servidor se puede realizar de dos formas distintas: mediante conexiones o mediante datagramas. Un protocolo orientado a conexión se basa en una conexión establecida en la que queda fijada el destino de la conexión desde el momento de abrirla (open). Esta conexión permanecerá hasta que se cierre, como ocurre con una llamada telefónica. Un protocolo de datagramas, por el contrario, envía mensajes indivi-

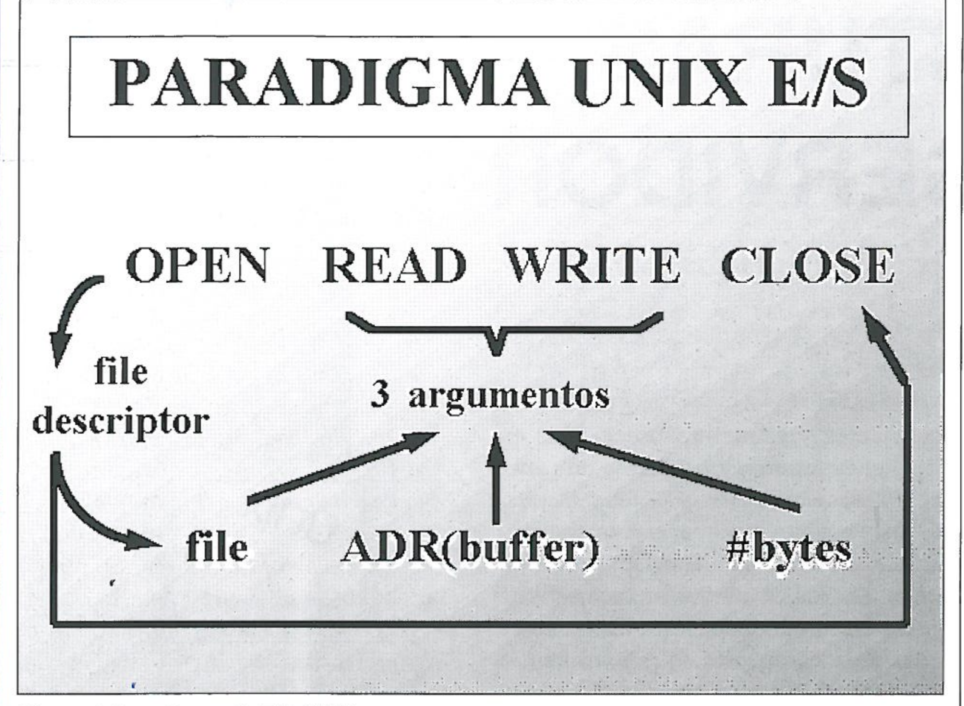


Figura 1. Paradigma de E/S UNIX.

duales con el destino grabado en cada uno de ellos, como en una comunicación a través de correo postal, donde no son necesarias las fases de establecimiento y liberación de conexión alguna.

LA ABSTRACCIÓN SOCKET

Para definir una interfaz de protocolos que permitiera una comunicación abierta entre procesos, la variante BSD de UNIX definió un IPC (Inter Process Communications) siguiendo el paradigma E/S de UNIX basado en 4 primitivas que se presentó el mes anterior y se puede observar en la figura 1. Así pues, se define la idea de socket como un punto abstracto de comunicación que se puede abrir (1) y cerrar (2) y sobre el que se puede leer (3) y escribir (4) como si se tratara de un fichero. De esta forma, un programa de aplicación utilizará un socket como un "canuto de comunicación" y lo tratará exactamente igual que haría con un fichero, con la diferencia de que lo que en él escriba será enviado a otro proceso en su propia máquina o en otra a miles de kilómetros y lo que de él lea estará generado por ese proceso interlocutor. Con ello, el sistema de comunicaciones UNIX queda perfectamente compacto y consistente con su filosofía E/S. Aunque la rama SYSTEM V define su propio IPC, el mecanismo de sockets de Berkeley se ha convertido en el estándar de facto incluso fuera del mundo UNIX por lo que se encuentra disponible en todas las versiones de UNIX (incluida Linux).

Cuando un programa de aplicación desea generar un socket, lo solicita al S.O. a través de una llamada al sistema y éste devuelve un entero de referencia al socket de forma análoga a lo que ocurre durante la apertura de un fichero. La diferencia entre un descriptor de fichero y un descriptor de socket consiste en que el S.O. enlaza (binds, en inglés) inmediatamente al primero con un fichero mientras que, en el caso del segundo, crea una referencia a socket sin generar realmente una comunicación con ningún destino. De esta forma, se permite al programa de aplicación que utilice el mismo socket para comunicarse con diferentes destinos enlazándolo de forma explícita con los mismos (bind) cuando más le conviene.

Siempre que es posible, se accede al socket igual que a un fichero a través de las llamadas write y read. De hecho, la integración es tan grande que el S.O. localiza los descriptores de socket y de fichero de entre el mismo conjunto.

LA GESTIÓN INTERNA

Como se estudió en el artículo correspondiente al sistema de ficheros, cada proceso tiene asociada una tabla de descriptores de fichero. Cuando un proceso UNIX abre un fichero, el S.O. coloca un puntero a la estructura del fichero en esa tabla de descriptores y devuelve un descriptor que será el que el programa de aplicación deberá recordar para hacer referencia al fichero y que el S.O. utiliza como índice en la tabla. Como se ha comentado en el punto anterior, un descriptor de socket no sólo es conceptualmente idéntico a un descriptor de fichero sino que, incluso, el S.O. coloca ambos en la misma tabla. De este modo, un programa de aplicación no podrá nunca disponer del mismo número como descriptor de socket y de fichero. La figura 2 refleja de forma gráfica estos conceptos.

CREACIÓN DE SOCKETS

Para realizar una comunicación, tanto el cliente como el servidor deberán crear un socket a través del cual comunicarse. Para la creación del mismo en el mundo UNIX se utiliza la llamada al sistema socket que se corresponde con la primitiva open del paradigma de E/S. La llamada responde a la forma

descriptor = socket(af,tipo,protocolo)

donde af, tipo, protocolo y descriptor son enteros.

El IPC de Berkeley se diseño de forma general, no sólo para TCP/IP. De esta forma, se puede tratar con distintas familias de protocolos. Así, el entero af representa la familia de protocolos que será soportada por el socket. Por ejemplo, sockets de dominio Internet (familia TCP/IP) vendrán especificados por la familia AF_INET y sockets de dominio UNIX (internos al S.O. local e integrados en el sistema de ficheros) por la

La mayoría de las aplicaciones cliente-servidor se fundamentan en el uso de sockets

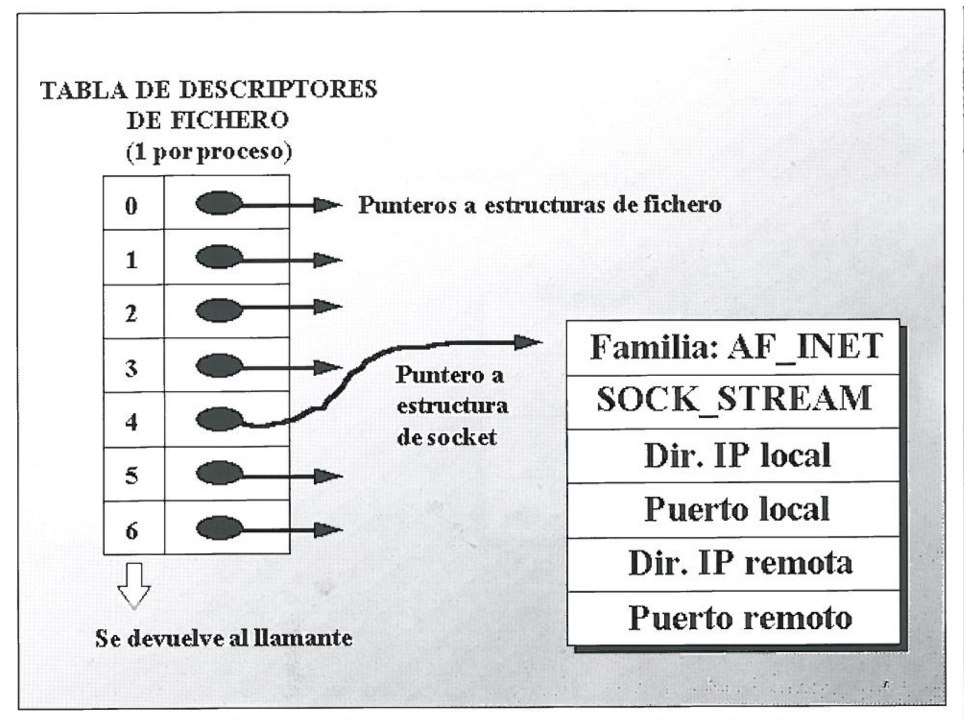


Figura 2. Los descriptores de socket y fichero pertenecen al mismo conjunto.

familia AF_UNIX. Con ello, se presenta una interfaz única al programa de comunicación independiente del sistema subyacente, ya quiera comunicar dos procesos locales o realizar una comunicación a través de la internet (figura 3).

Para no perder generalidad, en lugar de especificar un protocolo concreto, el entero tipo representa el tipo de servicio deseado. Este servicio puede ser orientado a conexión, SOCK_STREAM, o por datagramas (no orientado a conexión), SOCK_DGRAM.

Aunque no es necesario saber más para programar comunicaciones, ya que el mecanismo es transparente al usuario (en este caso, usuario programador), se comentará para los lectores que deseen profundizar más que, en ambos casos, el sistema operativo creará un punto de conexión en el protocolo de nivel 4 (nivel de transporte) correspondiente: en el caso de Internet (arquitectura TCP/IP) las conexiones serán soportadas por el protocolo TCP, fiable, y los datagramas por el protocolo UDP, no fiable.

El concepto de fiabilidad en arquitecturas de protocolos implica tres garantías: 1. No se perderán paquetes; 2. No llegarán paquetes duplicados; 3. El orden de recepción será el mismo que el de emisión. Si se emplea un protocolo fiable, el programa de aplicación quedará liberado de la responsabilidad de gestionar reordenaciones, pérdidas y duplicados. A cambio, el software de protocolos interno necesitará un mayor tiempo de proceso. En ocasiones, por la propia naturaleza del protocolo, será incluso irrelevante la pérdida, duplicado o desorden de algún mensaje, por ejemplo, en situaciones en que es importante el tiempo real, como los datos que puedan regir un ecualizador gráfico.

Existe otro tipo de socket, el socket crudo SOCK_RAW que puenteará al nivel de transporte y accederá directamente al nivel 3 (nivel de red): en el caso de internet, IP. En cualquier caso, la utilización de este tipo de socket se reserva a usuarios experimentados ya que suele ser utilizado por programas con privilegios para la definición de protocolos de bajo nivel. En la figura 4 se ilustra esta clasificación.

Por último, el entero protocolo selecciona el protocolo a utilizar en caso en que el nivel de transporte de la arquitectura de red que gestiona el sistema disponga de varios protocolos orientados conexión o varios no orientados a conexión.

ESPECIFICACIÓN DE LA DIRECCIÓN LOCAL

Como se ha comentado, la creación de un socket no supone su asociación a ninguna dirección, ni de origen, ni de destino. Desde el punto de vista del servidor, será necesario disponer de una dirección local en la que aceptar peticiones. En la arquitectura TCP/IP, una dirección del nivel de transporte (nivel 4) viene especificada por una dirección del nivel de red (nivel 3, IP) más un número de puerto. En la mayoría de los casos, no suele importar en que puerto ha de hallarse el servidor, por lo que se deja que éste sea escogido por el software de protocolos. En otras ocasiones, se trata de servidores "normalizados" que deberán atender en un puerto local específico. Así, por ejemplo, el servidor de SMTP (Simple Mail Transfer Protocol) para correo "escucha" siempre en el puerto 25 o el de HTTP (HyperText Transfer Protocol), el del famoso WWW, "escucha" siempre en el 80. Los primeros 1024 puertos son de uso reservado para root con objeto de servir de garantía al cliente de que realmente está enviando información al programa adecuado. Para asociar la dirección local a un socket se dispone de la llamada al sistema bind:

bind(socket,local_addr,addr_len)

donde socket es el descriptor de socket que nos ha devuelto la llamada socket, local_addr es una estructura sockaddr y addr_len es su tamaño. En la figura 5 se muestra la forma de la estructura sockaddr definida en <netinet/in.h>.

En el caso en que se esté tratando con sockets de dominio UNIX, que aparecerán como un fichero más en el sistema de ficheros, será suficiente indicar el nombre de éste:

bind(s,"/dev/sock",sizeof("/dev/sock")-1);

Para realizar una comunicación tanto el cliente como el servidor deberán crear un socket

CONEXIÓN CON DIRECCIONES DESTINO

Desde el punto de vista del cliente, será necesario realizar las peticiones a una dirección destino determinada en la que se encontrará esperando el servidor. A la hora de establecer direcciones de destino existen diferencias cuando se trata con sockets orientados a conexión (STREAM SOCKETS) o a través de datagramas (DATAGRAM SOCKETS). La asignación de la dirección de destino se realiza a través de la llamada connect:
connect(socket,dest_addr,addr_len)
análoga a bind. En el caso de un STREAM SOCKET, la utilización de connect es obligatoria y realiza una conexión sobre un protocolo de transporte fiable. En el caso de DATAGRAM SOCKETS, con los que no existe conexión, connect simplemente ahorra al programador tener que indicar en cada datagrama la dirección de destino y deja esta tarea para el software de red, por lo que su utilización es opcional.

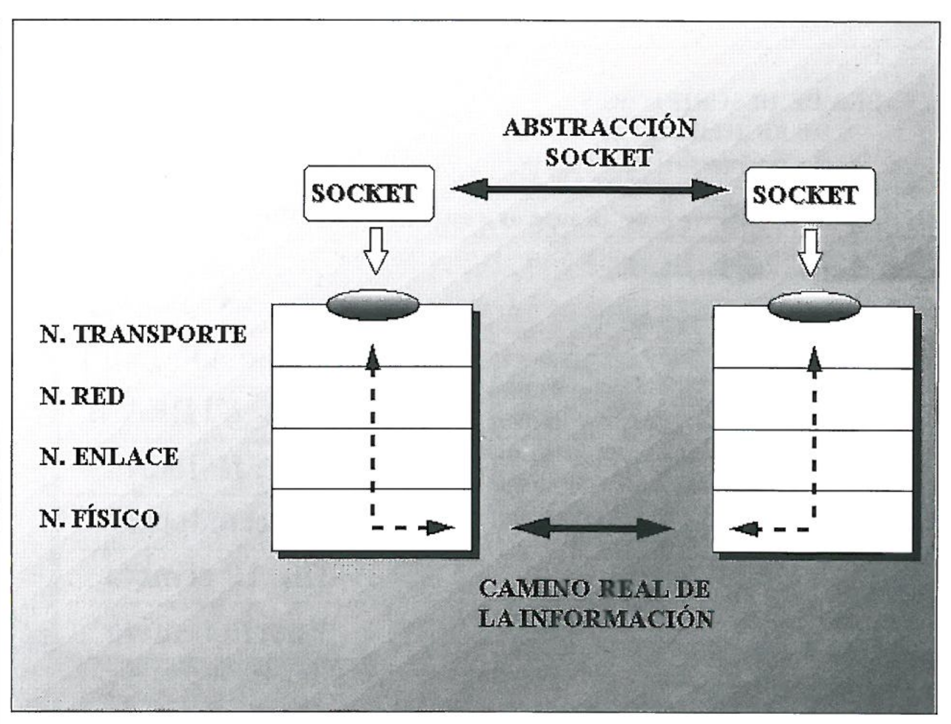


Figura 3. Abstracción socket: las aplicaciones "ven" una comunicación directa.

ESPECIFICACIÓN DE COLAS DE SERVIDORES

Una vez un servidor ha creado un socket orientado a conexión y éste ya dispone de dirección local en la que escuchar (asignada por el programador o por el sistema), es necesario indicar al S.O. el tamaño de la cola de peticiones a ese socket que se desea que mantenga. Esto se debe a que en el momento en que el servidor está procesando una llamada puede llegar otra. En un servicio de datagramas se podría simplemente rechazar (se admite que haya pérdidas); sin embargo, en un servicio fiable se indica al sistema operativo que guarde una cola mediante:
listen(socket,longitud)
donde longitud es el número de peticiones que aceptará la cola asignada al socket.

Un descriptor de socket nunca puede coincidir con uno de fichero en el mismo programa de aplicación

ADMISIÓN DE CONEXIONES

Suponiendo un servidor orientado a conexión, según lo visto hasta momento, se habrá creado un socket mediante socket, se le habrá asignado una dirección mediante bind y se habrá indicado al S.O. la cantidad de peticiones a esa dirección que debe mantener en cola mediante listen. Sin embargo, aún no se encontrará listo para aceptar estas peticiones. Para ello, se define la llamada accept, de la forma:
nuevo_socket=accept(socket,remote_addr,addrlen)
donde remote_addr indica la dirección por la que se desea atender. Esto se debe a que una máquina puede disponer de varias conexiones a varias redes y, por tanto, de varias direcciones. En caso de que se desee atender llamadas desde cualquier dirección, se dispone de la constante ADDR_ANY.

La primitiva accept es bloqueante si no se especifica lo contrario. Es decir, suspende la ejecución del programa hasta que se realiza una petición. En ese momento, el S.O. crea un socket nuevo devolviendo su descriptor en nuevo_socket y rellena las estructuras remote_addr y addr_len con los datos del cliente. De esta forma, el servidor podrá rechazar llamadas de clientes "indeseados". El socket original seguirá aceptando llamadas.

HERENCIA Y CIERRE DE SOCKETS

Al replicar un proceso mediante fork o reemplazarlo mediante exec, el S.O. el nuevo proceso hereda y mantiene el acceso a todos los sockets y ficheros abiertos del original. Si bien el S.O. mantiene una referencia del número de programas que acceden a un descriptor, su manejo es únicamente responsabilidad del programador. Para cerrar un socket, el sistema proporciona la llamada close, de la forma:
close(socket)

Internamente, el S.O. decrementa en una unidad la referencia de programas que acceden al descriptor al recibir una llamada close y únicamente cuando la referencia alcance el valor cero será cuando destruya el socket realmente. En cualquier caso, desde el punto de vista del programa de aplicación que realiza un close, el socket queda destruido irreversiblemente.

Si está tratando con sockets de dominio UNIX que, como se ha mencionado anteriormente, aparecen en el sistema de ficheros como un fichero más, no bastará con la llamada a close

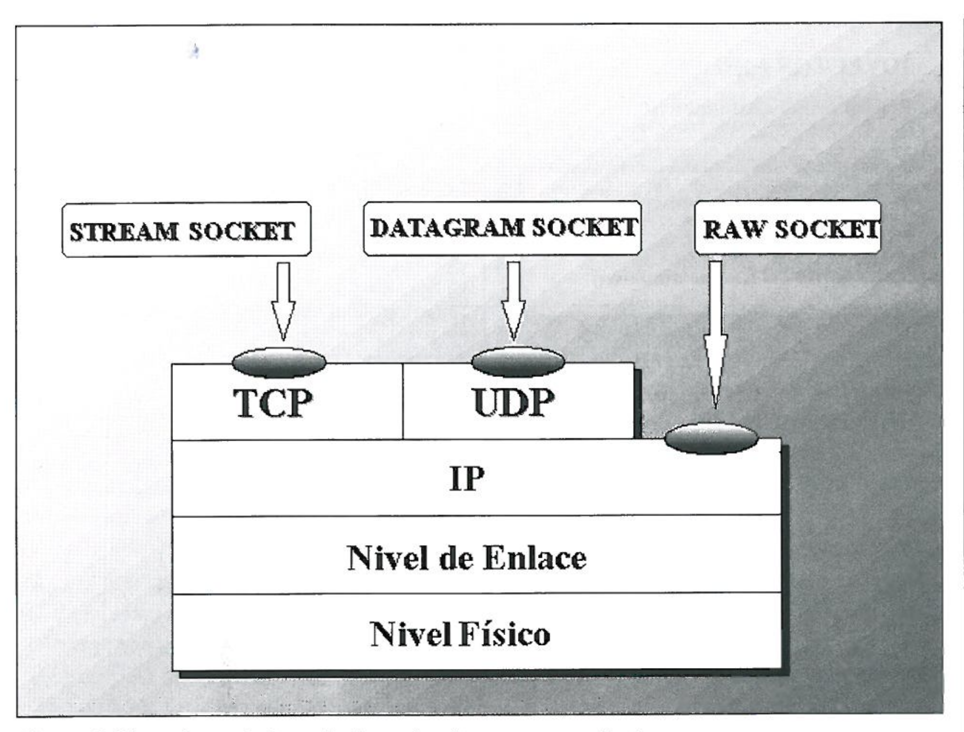


Figura 4. Tipos de sockets y niveles sobre los que se sustentan.

si no que, además, deberá llamarse a unlink (ver artículo del mes anterior) para eliminar su entrada de directorio.

También es posible cerrar únicamente una "mitad" del socket, la de lectura o la de escritura mediante la llamada shutdown:
shutdown(s,flag)
donde flag valdrá 0, 1 ó 2 para indicar cierre del lectura, escritura o ambas respectivamente.

RUTINAS DE CONVERSIÓN DE ENTEROS

La arquitectura TCP/IP define una representación estándar para los enteros binarios utilizados en las cabeceras de sus PDUs (Unidad de Datos de Protocolo) a la que denomina network byte order y que consiste en colocar primero el byte más significativo. Así pues, los enteros correspondientes, como por ejemplo, el puerto de protocolo que se utilicen en la programación de comunicaciones, deben ser expresados en este orden. Para ello, el IPC proporciona funciones de conversión del orden interno al red y viceversa para enteros de 16 y 32 bits. A saber: htons (host to network short), ntohs (network to host short), htonl (host to network long) y ntohl (network to host long). Si bien, el orden de representación de red puede coincidir con el de su máquina, es conveniente hacer uso siempre de estas funciones para hacer portables los programas.

ENVÍO DE DATOS A TRAVÉS DE SOCKETS

Para enviar información a través de sockets el S.O. facilita cinco llamadas al sistema. Tres, para comunicaciones orientadas a conexión: send, write y writev y dos, para servicios de datagramas: sendto y sendmsg.

1. write(socket,buffer,longitud)
donde en buffer se pasa un puntero al buffer con los datos a enviar y longitud es el número de bytes del mismo que se enviarán. No es necesario indicar dirección de destino ya que al utilizarse en comunicaciones orientadas a conexión, el socket estará conectado a una dirección remota a través de un accept o un connect. Como se puede apreciar, se trata de la misma primitiva empleada en el acceso a ficheros.

2. writev(socket,iovector,vertorlen)
es una llamada que hace uso de un recurso de algunos S.O. distribuidos o en red llamado gathering por el que no es necesario copiar el mensaje en bytes contiguos de memoria con el consiguiente gasto de tiempo que ello llevaría. En este caso, para identificar la información será necesario indicar la dirección de un array de tipo iovector que consiste en una secuencia de punteros a los bloques de datos así como la longitud de cada bloque (figura 6).

3. send(socket,mensaje,longitud,flags)
donde mensaje es la dirección de los datos y flags una serie de opciones sobre la comunicación para indicar, por ejemplo, datos fuera de banda o expedited data. El uso de estos flags implica que el programador de comunicaciones conoce las características del protocolo de transporte que dará servicio a su aplicación y escapan al contenido del articulo de este mes.

4. sendto(socket,mensaje,longitud,
flags,destino_addr,addrlen)
se utiliza para envíos en sockets no conectados (indica dirección destino).

5. sendmsg(socket,struct_mensaje,
flags)

donde struct_mensaje queda definido en la figura 7.

RECEPCIÓN DE DATOS A TRAVÉS DE SOCKETS

Análogamente, para la lectura de datos se dispone de otras cinco llamadas:

La comunicación entre aplicaciones puede ser orientada a conexión o mediante datagramas

1. read(desc,buffer,longitud)
2. readv(desc,iovector,vectorlen)
3. recv(socket,buffer,longitud,flags)
4. recvfrom(socket,buffer,longitud,
flags,origen_addr,addrlen)
5. recvmsg(socket,struct_mensaje,
flags)

que quedan definidas por sus homólogas de envío. La única aclaración a realizar es que la primitiva recvfrom, descarta aquellos datos que no quepan en

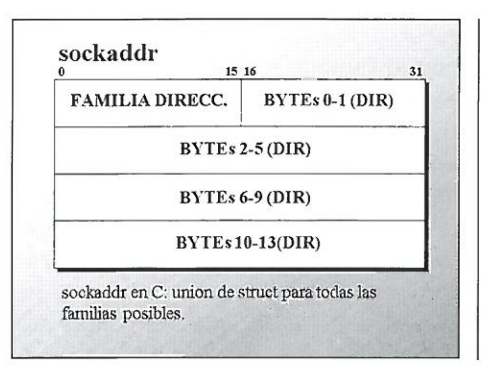


Figura 5. Estructura sock_addr.

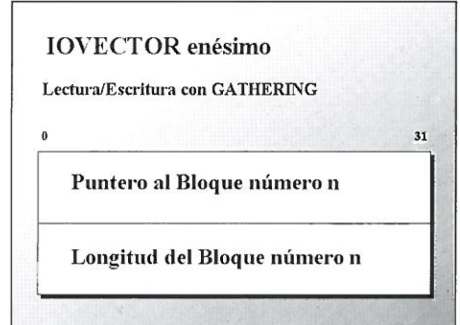


Figura 6. IOVECTOR n-simo.

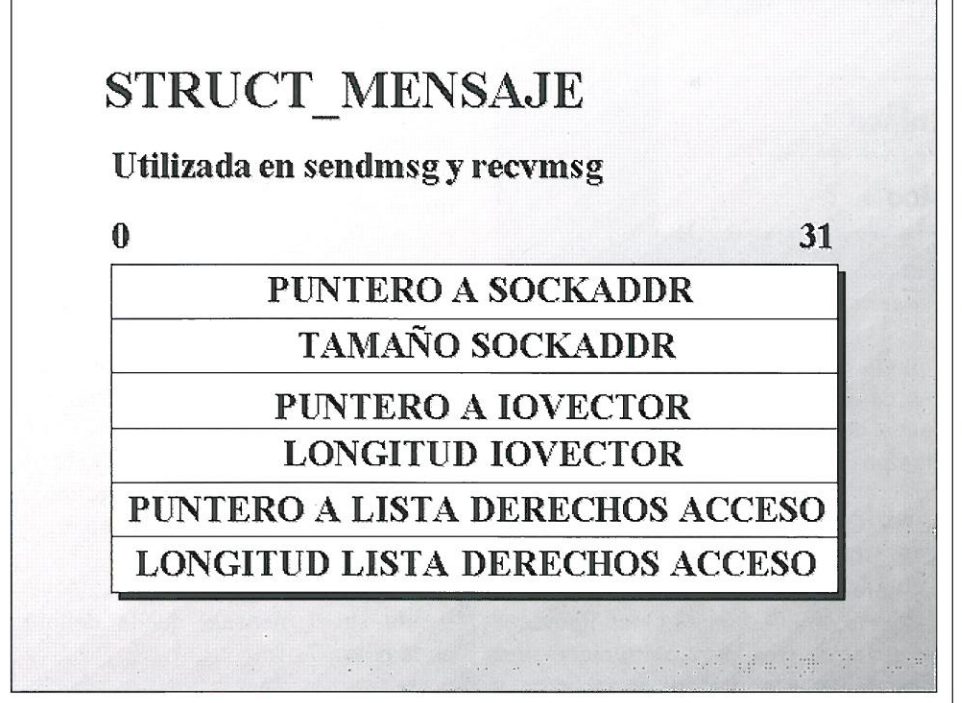


Figura 7. Estructura mensaje.

TABLA 1

```
#include <sys/types.h>
#include <sys/socket.h>

int socket (int domain, int type, int protocol);
int bind (int sockfd, struct sockaddr *my_addr, int addrlen);
int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
int accept (int s, struct sockaddr *addr, int *addrlen);

int send (int s, const void *msg, int len, unsigned int flags);
int recv (int s, void *buf, int len, unsigned int flags);

int sendto  (int s, const void *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen );
int recvfrom(int s, void *buf, int len, unsigned int flags, struct sockaddr *from, int *fromlen);

int sendmsg (int s, const struct msghdr *msg, unsigned int flags);
int recvmsg (int s, struct msghdr *msg, unsigned int flags);

#include <sys/socket.h>
int listen(int s, int backlog);

#include <sys/types.h>
#include <unistd.h>
ssize_t write(int fd, const char *buf, size_t count);
int read (int fd, char *buf, size_t count);

#include <unistd.h>
int close(int fd);

#include <sys/uio.h>
int readv (int filedes, const struct iovec *vector, size_t count);
int writev(int filedes, const struct iovec *vector, size_t count);
```

el buffer debido a que se trata de un servicio no fiable de datagramas.

OBTENCIÓN DE DIRECCIONES LOCALES Y REMOTAS

En algunos casos puede ser desconocida la dirección local de un socket, bien porque el socket haya sido heredado, bien porque el puerto (caso AF_INET) haya sido asignado por el S.O. Del mismo modo, el programa de aplicación puede desear conocer la dirección de un cliente. Para ello, el IPC proporciona dos llamada más:
getpeername (socket,dest_addr,addrlen)
getsockname(socket,local_addr,addrlen)
que proporcionan las direcciones remota y local respectivamente y donde dest_addr, local_addr y addrlen son punteros a direcciones que rellenará la función.

El IPC de Berkeley puede tratar con distintas familias de protocolos

CONCLUSIÓN

Se han presentado las herramientas básicas para la comunicación con sockets. Sin embargo, aún no serán suficientes en todos los casos. Si desea practicar por su cuenta inténtelo con sockets de dominio UNIX. Si no obtiene resultados, no desespere. Aún es necesario dotar de estructura a lo expuesto y presentar algunas herramientas más. Aproveche este mes para asimilar los conceptos teóricos que en el próximo número se complementarán con la realización de una aplicación cliente-servidor.

CONTACTAR CON EL AUTOR

Para cualquier clase de duda, aclaración, comentario, sugerencia o crítica, se anima a los lectores a que se pongan en contacto con el autor a través de carta a la redacción dirigida a la sección Correo del Lector de la revista o, preferiblemente, mediante:

E-Mail Internet: echeva@dit.upm.es
E-Mail Compuserve: 100646,2456
WWW:
http://highland.dit.upm.es:8000
ftp: highland.dit.upm.es

LA TRANSFORMACIÓN RÁPIDA DE FOURIER

Juan Ramón Lehmann

En anteriores artículos se ha visto la transformada de Fourier, sin embargo, el principal impedimento para usar la DFT es su alto coste en tiempo. En este articulo se verá la implementación de la FFT (Fast Fourier Tranformation) o lo que es lo mismo, la transformada rápida de Fourier que solventa este problema.

En 1942 Danielson y Lenczos aportaron una solución recursiva para la DFT. Ellos comprobaron que una DFT de longitud Nâ podía ser reescrita como la suma de dos DFT, cada una de longitud N/2, donde N es un entero potencia de 2. La primera DFT hace uso de los puntos pares de la secuencia original N, la segunda hace uso de los impares. La siguiente ecuación sirve de prueba de lo anteriormente mencionado.

$$
\begin{aligned}
F_{n+N} &= \sum_{k=0}^{N-1} f_k e^{-i2\pi(n+N)k/N} && \text{(A1.2.5)} \\
&= \sum_{k=0}^{N-1} f_k e^{-i2\pi nk/N} e^{-i2\pi Nk/N} \\
&= \sum_{k=0}^{N-1} f_k e^{-i2\pi nk/N} \\
&= F_n
\end{aligned}
$$

La ecuación A1.2.1 coincide con la definición original de la DFT. La suma expresada en A1.2.1 como dos sumas más pequeñas, se corresponde con los puntos pares e impares respectivamente. Para un correcto acceso a los datos, el índice es intercambiado de k a 2k+1, y el límite superior se vuelve (N/2)-1. Esos cambios están reflejados en A1.2.3, donde ambas sumas son expresadas en una forma equivalente a una DFT de longitud N/2. Se simplificará la notación (Ec. A1.2.4) con objeto de hacerla más llevadera a lo largo del artículo.

Fen será el enésimo componente de la transformada de Fourier de longitud N/2 formada por los puntos pares de la f, mientras que Fon corresponden a los impares.

Danielson-Lanczos Lemma presentan una solución que pasa por el dicho "divide y vencerás". Se puede observar que la dificultad de calcular Fn es reducida a calcular dos sumas independientes, Fen y Fon.

Simplificando, podemos observar que Fn es periódica en la longitud de la transformación, esto es, una DFT de longitud N, Fn+N=Fn, como podemos observar en la Imagen 2.

$$
\begin{aligned}
F_n &= \sum_{k=0}^{N-1} f_k e^{-i2\pi nk/N} && \text{(A1.2.1)} \\
&= \sum_{k=0}^{(N/2)-1} f_{2k} e^{-i2\pi n(2k)/N} + \sum_{k=0}^{(N/2)-1} f_{2k+1} e^{-i2\pi n(2k+1)/N} && \text{(A1.2.2)} \\
&= \sum_{k=0}^{(N/2)-1} f_{2k} e^{-i2\pi nk/(N/2)} + W^n \sum_{k=0}^{(N/2)-1} f_{2k+1} e^{-i2\pi nk/(N/2)} && \text{(A1.2.3)} \\
&= F_n^e + W^n F_n^o && \text{(A1.2.4)}
\end{aligned}
$$

De lo cual deducimos:

Por tanto [illegible] $(n+N/2)) = -\cos(2\pi n/N) \quad 0 \le n < N/2$ (A1.2.7)

[illegible] $(n+N/2)) = -\sin(2\pi n/N) \quad 0 \le n < N/2$

Therefore,

$$W^{n+N/2} = -W^n \quad 0 \le n < N/2 \qquad \text{(A1.2.8)}$$

Esto permite que N/2 valores de Fen y Fon, generen los N números necesitados para Fn. Una simplificación similar existe para el Wn factor de la ecuación A1.2.4. Desde que W

$$
\begin{aligned}
F_n &= F_n^e + W^n F_n^o && 0 \le n < N/2 && \text{(A1.2.9)} \\
F_{n+N/2} &= F_n^e - W^n F_n^o && 0 \le n < N/2
\end{aligned}
$$

es periodo de N, los primeros N/2 valores pueden ser usados con objeto de generar el resto de los N/2 valores de acuerdo con la siguiente relación:

CONTROL DE FLUJO MARIPOSA

La ecuación A1.2.9 puede ser representada por un control de flujo mariposa como la imagen 5, donde el signo negativo en +-Wn se eleva en el cálculo de Fm+N/2. Los transcursos a lo largo de las flechas, representan factores multiplicativos aplicados a los datos de entrada, mientras que las intersecciones de las flechas, representan sumas. Por simplificación, tomaremos la opción (b) que es equivalente a la (a). La complejidad de cálculo de la parte real y la parte imaginaria de su parte compleja, está reflejada por los subíndices r e i respectivamente.

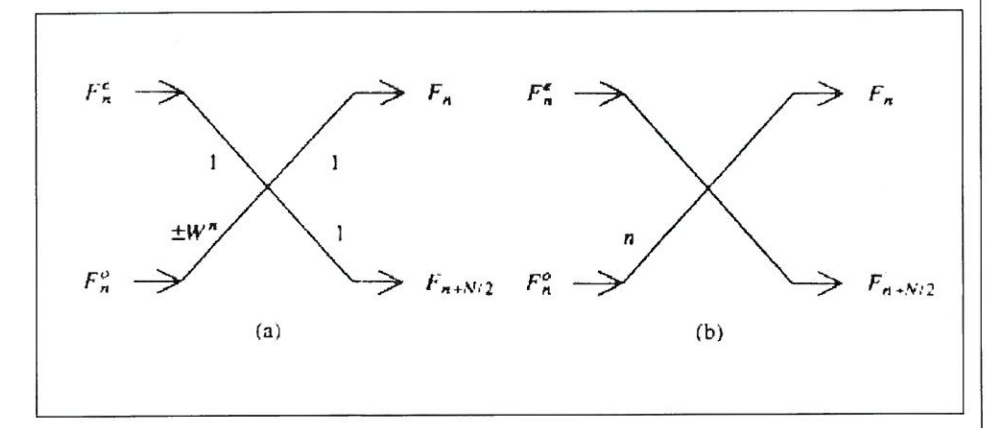
La parte real e imaginaria de Wnh son wrhr-wihi y wrhi+wihr respectivamente.

EL INTERCAMBIO DE PUNTOS

La forma recursiva aportada por Danielson-Lanczos Lemma se demuestra fácilmente en el siguiente diagrama. Se considerará la lista f como 8 números complejos etiquetados desde f0 hasta f7 en el diagrama. Para poder reasignar las entradas de la lista con los coeficientes Fn, se deberá evaluar Fen y Fon. Como resultado, dos nuevas listas son generadas conteniendo los puntos pares e impares de dicha lista respectivamente. Aplicando el mismo procedimiento a las dos listas generadas (las nuevas), son realizadas sucesivas subdivisiones hasta que se alcanza el árbol en su nivel más bajo, dejando solamente un elemento por cada lista.

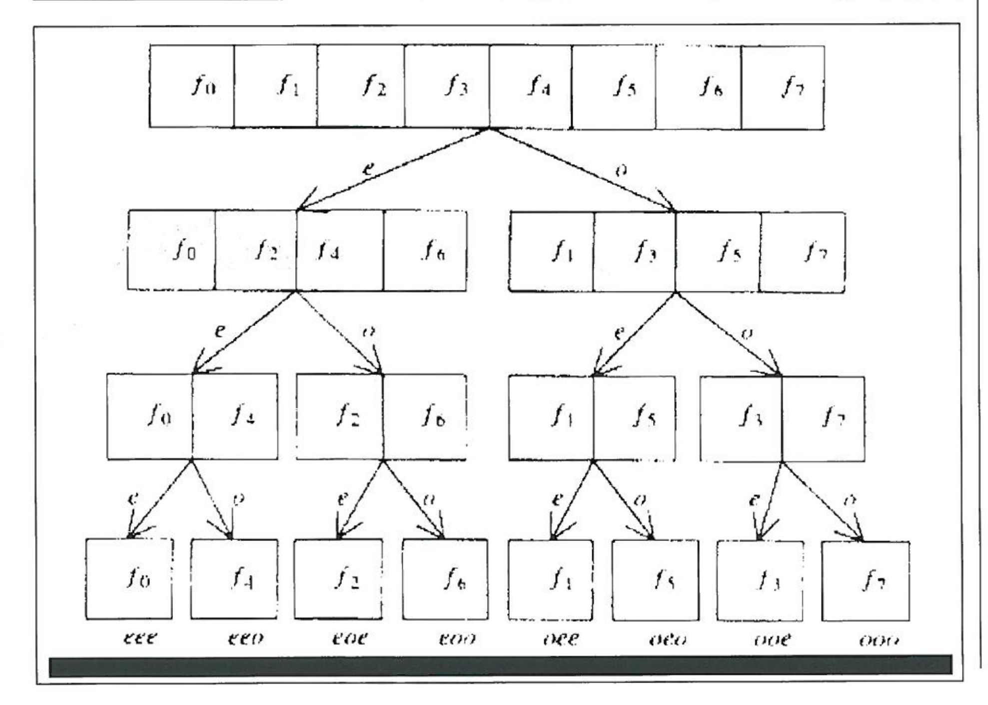
Una vez alcanzado este punto, comenzamos a desandar el camino árbol arriba, construyendo los coeficientes usando la ecuación A1.2.9. El diagrama 2 visualiza este proceso haciendo uso de un control de flujo mariposa. Nótese que las líneas en negrita son usadas para señalar los límites de las listas. Empezando con el primer elemento DFT, el primer punto es evaluado primero. Nótese que el mismo permanece inalterable.

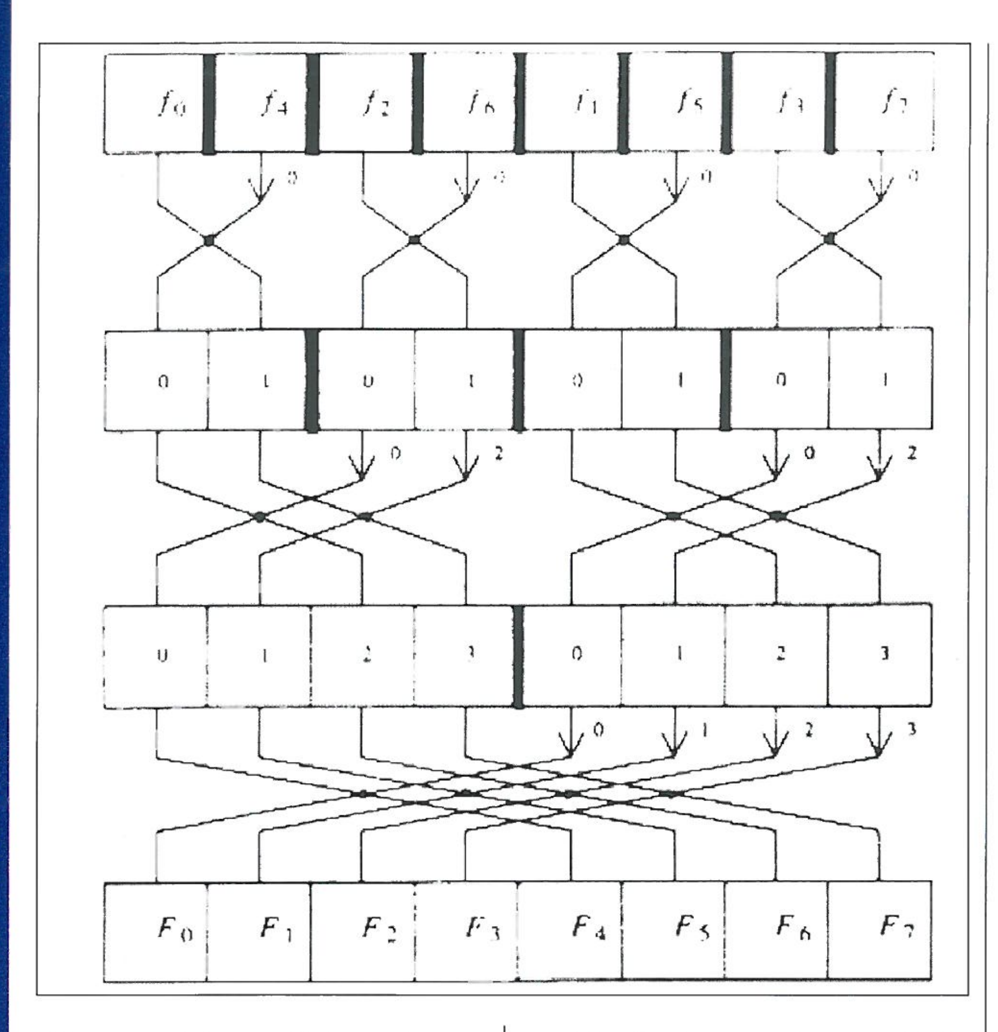
La transformación del 2º punto usa el resultado de la transformación del 1º. Los cuatro siguientes los resultados del segundo, y así sucesivamente. En este caso particular, N vale 4 y el exponente de W desde 0 a (N/2)-1 ó 1. En el diagrama 3 se asume que



$$g = F_n^e$$
$$h = F_n^o$$
$$w_r = \cos(-2\pi n/N)$$
$$w_i = \sin(-2\pi n/N)$$

$$\begin{aligned} F_n &= g + W^n h \qquad \text{(A1.2.10)} \\ &= [g_r + i g_i] + [w_r + i w_i][h_r + i h_i] \\ &= [g_r + i g_i] + [w_r h_r - w_i h_i + i w_r h_i + i w_i h_r] \\ &= [g_r + w_r h_r - w_i h_i] + i[g_i + w_r h_i + w_i h_r] \end{aligned}$$





N=8 para el factor W. Los números están normalizados siguiendo este baremo. Para la transformación de 4 puntos, el exponente de 0 y 1 (suponiendo N=4) resulta ser 0 y 2 para compensar el valor de N=8, el último paso en la transformación de los 8 puntos. En general, se combinan un par de los puntos adyacentes para obtener 2 puntos transformados, entonces se combinan los pares adyacentes de los adyacentes para conseguir 4, y subsiguientes hasta que la primera y segunda mitad del total de los datos han sido combinados en la transformación final.

existen otros muchos sistemas de realizarla como podrían ser el de Cooley-Tukey, Cooley-Sande, etc...

SEUDOCÓDIGO

Este algoritmo es fácilmente programable, en el presente artículo se incluye el Seudocódigo que hace posible la implementación de la FFT.

```
Function FFT(N,f)
if (N=2)
* reemplazar f0 por f0+f1 y f1 por f0-f1
return;
else
* Definir g como una lista de todos los puntos de f, la cual tiene los pares y h como una lista con los impares.
* FFT(N/2,g)
* FFT(N/2,h)
(1)* Reemplazar fn por gn+Wnhn => for n=0 a N-1
```

El Seudocódigo aquí explicado, es invocado con dos argumento: N y f, N es el número de puntos a pasar en el array f. Ya que N>2, f es dividido en dos partes g y h. El array g almacena los puntos pares de f, mientras que h almacena los impares. La transformada de esas dos listas es calculada invocando la función FFT con los parámetros g y h con valor N/2 (longitud de array). La recombinación de las dos listas DFT ocurre en (1) de acuerdo con la ecuación A1.2.4.

Las subdivisiones continúan hasta que N=2, en este punto, el exponente de W es truncado a 0. Como W0 es 1 (cualquier número elevado a 0 =1), no hay necesidad de realizar más multiplicaciones ya que los resultados se obvian directamente (línea 2).

Volviendo al punto (1), ahorraremos tiempo de computación usando los N/2 elementos disponibles en g y h para generar los N números requeridos. Para simplificar el seudocódigo, se incluye el fuente comentado en C que realiza el (1). Nótese que todas las variables excepto N, N2 y n son de tipo double.

```
ang=0; /* inicializo el ángulo */
inc=-6.2831853/N; /* incremento de ángulo: 2pi/N */
N2=N/2;
for(n=0;n<N2;n++)
{
wr=cos(ang); /* parte real de Wn */
wi=sin(ang); /* parte imaginaria de Wn */
ang+=inc;
a=wr*hr[n]-wi*hi[n];
fr[n]=gr[n]+a;
fr[n+N2]=gr[n]-a;
a=wi*hr[n]+wr*hi[n];
fi[n]=gi[n]+a;
fi[n+N2]=gi[n]-a;
}
```

CONCLUSIÓN

Existen varios algoritmos para implementar la FFT. En este artículo se ha comentado uno de ellos, pero existen otros muchos sistemas de realizarla como podrían ser el de COOLEY-TUKEY, COOLEY-SANDE, etc... Estos son públicos y fácilmente accesibles en los forums de "graphics algorithms" de Internet.

Con esta entrega finaliza esta serie dedicada a la transformación de Fourier dentro del capítulo dedicado a el tratamiento digital de la imágen.

LA LENTE

Pedro Antón

En esta nueva entrega del curso para aprender a realizar una demo pasamos a estudiar un nuevo efecto: la lente, este efecto simula el paso de una bola de cristal sobre la pantalla.

La idea principal es transformar una porción de dibujo para simular el efecto que produciría una bola de cristal al pasar por encima de una imagen. Para ello, se tomará el cuadrado más pequeño posible, que contendrá todos los pixels afectados por la transformación. Se precalculará la transformación, por lo tanto es necesario mantener una serie de constantes fijas: ancho de la lente, radio de la lente, y distancia del plano que corta la imagen al centro de la esfera. De esta manera el numero de pixels que se ven afectados por la transformación será el ancho de la lente por dos.

Serán necesarios tres arrays para generar la transformación:

*LenteOrg [1..Ancho*Ancho] of Byte;*

*LenteTrans [1..Ancho*Ancho] of Word;*

LenteDes [1..Ancho*Ancho] of Byte;

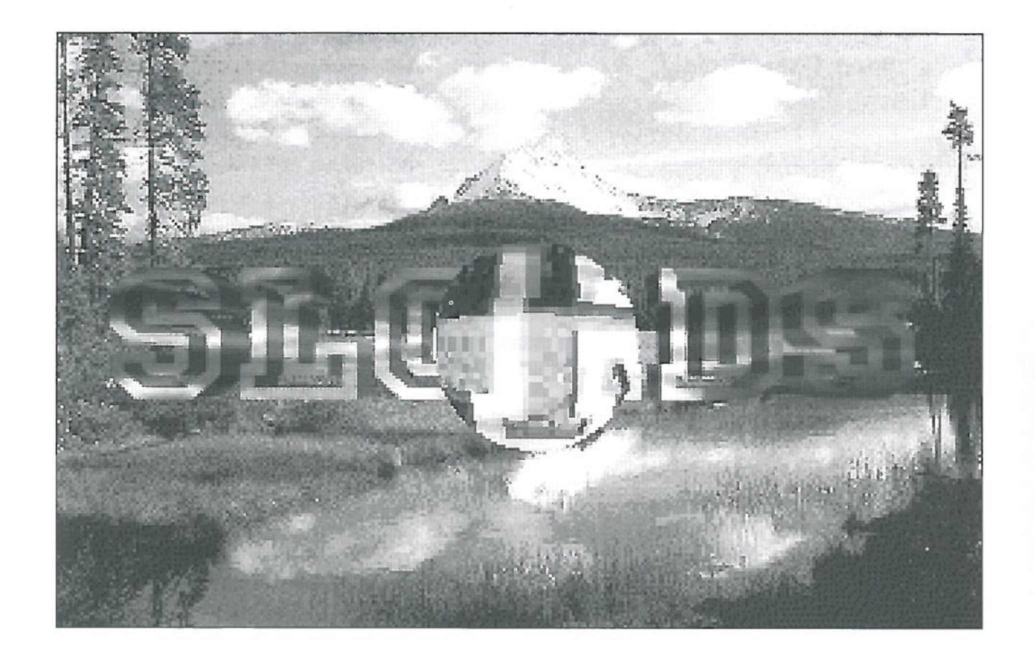
El primer paso es copiar la imagen a transformar al array LenteOrg, de tal forma que contenga una copia exacta de la zona de memoria a transformar. Al referirse a una zona de memoria queda claro que se transforma un buffer que puede ser o no la imagen existente en la memoria de vídeo, si la zona de memoria que se transforma es una modificación de la memoria de vídeo, se puede conseguir el efecto que genera Ispania en su demo PUAJJJ.

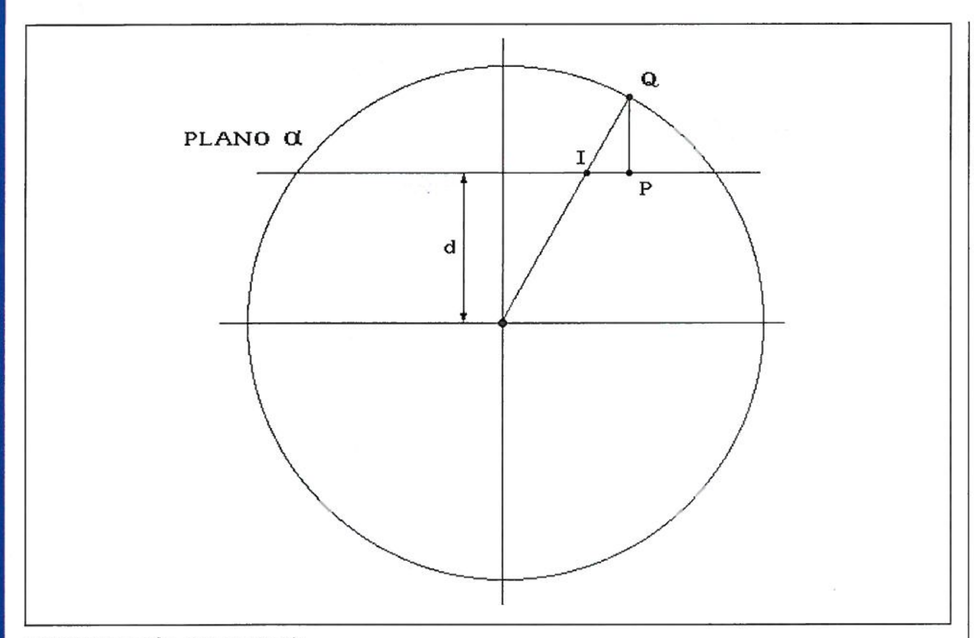


A continuación se procederá a transformar el array que contiene los datos a transformar, LenteOrg. Se determinará el nuevo valor para cada pixel que se almacenará en LenteDes.

Para determinar este nuevo valor, se usará un tercer array LenteTrans. Este array es constante y se precalcula con anterioridad, esto quiere decir que no es necesario crear el array cada frame. En el ejemplo adjunto, se calcula al inicio de la ejecución del programa.

Una vez calculada la imagen transformada se vuelca a la zona de memoria deseada y el proceso queda concluido. Es interesante destacar la parte del





Determinación del punto Q.

programa que restaurará la zona anteriormente escrita. Si el diámetro de la lente es menor que el ancho de la imagen a transformar se puede mover la lente usando incrementos de x y de y correspondientes a la diferencia entre ambos valores divido entre dos, sin necesidad de restaurar la antigua imagen transformada. De lo contrario se debe escribir en una zona de memoria virtual, pudiendo realizar más efectos, como la superposición de dos lentes. Nótese que la escritura directa sobre la memoria de vídeo, creará un molesto parpadeo.

El problema, por lo tanto, de este efecto es la creación de la matriz de transformación. Esta matriz puede generar tantos efectos como al lector se le ocurra.

Sea *Size=Ancho*Ancho*. La matriz generada a continuación, no transformará la imagen en absoluto:

FOR Cnt=1 TO Size DO LenteTrans[Cnt]=Cnt

Ya que el proceso para crear la matriz de destino es el siguiente:

FOR Cnt=1 TO Size DO LenteDes[Cnt]= LenteOrg[LenteTrans[Cnt]]

Por lo tanto, a continuación se procederá a crear la matriz de transformación adecuada, para conseguir el efecto de una lente, recurriendo, como viene siendo habitual a las matemáticas.

ANÁLISIS MATEMÁTICO DEL EFECTO LENTE

Una vez es necesario el uso de las matematicas.

Sea una esfera S, cortada por un plano alfa, esto define un circulo C en el plano alfa, siendo Rc el radio del circulo resultante, recordando el teorema de Pitágoras, el radio del círculo al cuadrado mas la distancia del centro de la esfera al plano alfa al cuadrado es el radio de la esfera al cuadrado.

Para variar el enfoque angular y mantener el tamaño de la imagen, se debe dejar constante Rc, como una lente es circular, si la superficie a transformar es de dimensiones RadioLente*RadioLente y, por supuesto el radio es RadioLente, la distancia del plano alfa al centro de la esfera determinará el enfoque angular de la lente, por lo tanto sabido Rc, que se mantendrá constante y la distancia del plano alfa al centro de la esfera, se calculará el radio de dicha esfera.

Sea la zona a transformar de dimensiones Ancho*Ancho y el radio de la lente a crear RadioLente, se elegirá una distorsión angular fijando la distancia del plano alfa al centro de la esfera, y se calculará el radio de dicha esfera. Ahora se transformará cada pixel del gráfico original, previamente almacenado en el array LenteOrg.

Sea P el punto que se va a transformar, si este punto se encuentra fuera del circulo C, el punto no será transformado, lo cual quiere decir que LenteTrans[P]=P. O definiendo los puntos en coordenadas X e Y *LenteTrans[X+Y*Ancho]=X+Y*Ancho*. Por el contrario si el punto se encuentra dentro del circulo, este punto será transformado.

El efecto que produce una lente es el de proyectar los puntos desde el centro de la esfera hacia su superficie. Es decir, dado un punto P determinar el punto Q, donde el punto Q es la proyección del punto I sobre el plano alfa. Para conocer donde se proyectará el punto P en la circunferencia formada por la intersección del plano alfa con la esfera S, se realizan los pasos indicados en la figura.

Se calcula el punto donde corta el punto P a la esfera, en la figura Q, uniendo posteriormente dicho punto con el centro de la esfera, donde esa línea corte al plano alfa, será el punto transformado de P. Es decir *LenteTrans[P]=I*

IMPLEMENTACIÓN DE LA MATRIZ DE TRANSFORMACIÓN

El problema de la matriz de transformación es como implementarla:

El ancho de la imagen a transformar, el radio de la lente y la distancia del plano alfa al centro de la esfera son constantes. Considerando el origen del sistema de coordenadas en el centro del circulo C, el eje X e Y estarán incluidos en le plano alfa.

La zona a transformar estará comprendida entre los puntos:

(-Ancho/2,Ancho/2) x (Ancho/2,-Ancho/2)

El eje Z será situado con la parte positiva en la dirección inversa al centro, luego el centro de la esfera en este sistema de coordenadas será (0,0,-D), con lo cual la ecuación de la esfera es:

$$x^2 + y^2 + (z+d)^2 = R_s^2$$

Como el radio de la esfera al cuadrado es la suma del cuadrado del radio de la lente y el cuadrado de la distancia del plano al centro de la esfera, se obtiene:

$$x^2 + y^2 + (z+d)^2 = R_c^2 + d^2$$

Los puntos que serán transformados cumplen (Px,Py,0), luego:

$$P_x^2 + P_y^2 < R_c^2$$

Las coordenadas del punto pueden ser encontradas por mediación del sistema de ecuaciones:

$$x = P_x$$
$$y = P_y$$
$$x^2 + y^2 + (z+d)^2 = R_c^2 + d^2$$

Donde la solución es: Qx=Px, Qy=Py y para Qz se obtiene:

$$P_x^2 + P_y^2 + Q_z^2 + 2dQ_z + d^2 = R_c^2 + d^2$$
$$Q_z^2 + 2dQ_z - R_c^2 + P_x^2 + P_y^2 = 0$$
$$D = d^2 - (P_x^2 + P_y^2 - R_c^2)$$
$$\text{como } R_c^2 > P_x^2 + P_y^2 \text{ se obtiene } Q_z = 1 \pm\sqrt{D}$$

Se obtienen dos soluciones ya que la línea corta a la esfera en dos puntos, pero tomando Qz>0, se obtienen las coordenadas de

$$(Q_x, Q_y, Q_z) = (P_x, P_y, 1+\sqrt{D})$$

Finalmente se calcula el punto I mediante:

$$\left.\begin{aligned} x &= \frac{(z - S_z)}{(Q_z - S_z)}(Q_x - S_x) + S_x \\ y &= \frac{(z - S_z)}{(Q_z - S_z)}(Q_y - S_y) + S_y \end{aligned}\right\} \text{Linea desde el centro de la esfera al punto Q}$$
$$z = 0 \qquad \text{Plano } \alpha$$

Hallando finalmente:

$$I_x = \frac{d}{Q_z + d} Q_x$$
$$I_y = \frac{d}{Q_z + d} Q_y$$
$$I_z = 0$$

Conocidas las coordenadas de Q, que pueden ser determinadas si las coordenadas de P son conocidas, y conocido D, se puede calcular I. Por tanto se puede decir que:

*LenteTrans [(Ancho/2+Px) + (Ancho/2+Py) * Ancho] = (Ancho/2+Ix) + (Ancho/2+Iy) * Ancho*

Siempre que Px, Py, estén dentro de la circunferencia de radio RadioLente.

CREACIÓN DE LA MATRIZ DE TRANSFORMACIÓN

Se creará una lente menor que la zona a transformar.

Creación de la matriz de transformación

```
Procedure MakeLenteTrans;
Const
 CentroX  = Ancho div 2;
 CentroY  = Ancho div 2;
 CtePlano  =         30;
Var
 CntDatosX : Integer;
 CntDatosY : Integer;
 Radio     : Integer;
 D,Qz        : Integer;
 Ix,Iy      : Integer;
Begin

 For CntDatosY := -(Ancho div 2) to (Ancho div 2)-1
do
  Begin
    For CntDatosX := -(Ancho div 2) to Ancho div 2
do
    Begin
Radio:= Round (Sqrt
(Sqr(CntDatosX)+Sqr(CntDatosY)));
         If Radio>RadioLente then
         Begin
LenteTrans [CentroX+ CntDatosX+ ((CentroY+
CntDatosY)*Ancho)]:=
CentroX+CntDatosX+((CentroY+CntDatosY)*
Ancho);
    End
         else
Begin
D:= Abs(Sqr(CtePlano)- Sqr(CntDatosX)
Sqr(CntDatosY)+Sqr(RadioLente));
Qz:=Round (1+Sqrt(D));
Ix:=(CtePlano*CntDatosX) div (Qz + CtePlano);
Iy:=(CtePlano*CntDatosY) div (Qz + CtePlano);
LenteTrans [ CentroX+ CntDatosX+ ((CentroY+
CntDatosY)* Ancho)]:=(Ancho div 2)+Ix+
(((Ancho div 2)+Iy)*Ancho);
          End;
   End;
 If WriteText then write('.');
 End;
End;
```

Para crear la matriz de transformación, se procederá como acaba de explicar. No obstante, destacar que existe una diferencia de diez pixels entre el diámetro de la lente a crear y el ancho de la imagen a transformar. Esto es así, para no tener que restaurar la imagen en desplazamientos de la lente de hasta 5 pixel por frame. Si la zona que se coge del buffer para transformar tiene una zona que no se transforma, esta zona, irá restaurando la imagen anteriormente expuesta.

LA VELOCIDAD

Este efecto consume muy poco tiempo de CPU.

Procedimientos nuevos de la unidad DEMOVGA

```
PROCEDURE ReadRaw (Var
DestinoPtr:Pointer;FileName:String); Var
    FFile : File;
Begin
    Assign (FFile,FileName);
    Reset (FFile,1);
    BlockRead(FFile,DestinoPtr^,64768);
    Close (FFile);
End;
PROCEDURE Copy64K
(SegOrg,SegDes:Word);assembler;
Asm
    push ds
    mov ax,SegOrg
    mov ds,ax
    mov ax,SegDes
    mov es,ax
    xor di,di
    xor si,si
    mov cx,32000
    rep movsw
    pop ds
End;
```

Creación de la matriz de transformación.

Generar la matriz de transformación es lento, realmente hay que realizar un gran número de cálculos, y puede ser un problema. Pero, el tamaño de dicha matriz, es relativamente pequeño, luego es posible, generarla en un fichero, para después incluirla como constante, evitando así el tiempo muerto, no obstante, como ocurre con otros efectos, se puede usar ese tiempo como tiempo de exposición de alguna imagen, que haya creado el grafista del grupo. El resto del efecto, es realmente rápido, copiar, transformar y poner una zona de memoria es algo que puede ser altamente optimizado, consiguiendo grandes velocidades de proceso. En el ejemplo que acompaña a este articulo, el proceso de captura del trozo a transformar y el de exposición del trozo transformado, son rápidos. Sin embargo, es posible optimizar aún más, si el efecto lo requiere. Cabe destacar la transferencia de datos usando REP MOVSW, para lo cual, previamente se debe comprobar si el numero de datos a transferir es par o impar. Para dividir por dos, bastará con rotar un bit a la derecha, en caso e existir acarreo, el numero es impar, luego se debe poner un dato más. El lector, habrá notado que esto, puede ser optimizado, usando valores pares para el ancho de la imagen a transformar.

EL BUFFER

Es imprescindible usar un buffer en este efecto.

El efecto está basado en movimientos de memoria, luego al ser los accesos a vídeo más lentos, como se ha venido recalcando a lo largo del curso, se deben evitar, en la medida de lo posible. Si la zona a transformar se encuentra en la zona de memoria correspondiente a la memoria de vídeo, la velocidad disminuirá. Además existe un problema añadido, el parpadeo. Si se trabaja directamente sobre la memoria de vídeo, habrá que restaurar la zona modificada antes de capturar la nueva imagen a transformar, si esto se realiza correctamente, se puede llegar a conseguir que no se note demasiado, pero el molesto parpadeo, es inevitable.

Como cosa curiosa, se explicará el uso de dos imágenes, con ligeras modificaciones. Sea una imagen inicial, por ejemplo, un dibujo con caricaturas de los componentes del grupo. Esta imagen será directamente volcada a la memoria de vídeo, mientras se genera la matriz de transformación, permaneciendo estática unos segundos. Y la segunda imagen, será una modificación de esta, por ejemplo, las caras de las caricaturas, pasan ahora a ser fotografías digitalizadas de los componentes del grupo. El buffer de donde tomaremos los datos a transformar, será este último, por lo tanto, cuando la lente se desplace sobre la cara de los componentes del grupo la imagen que aparecerá distorsionada será la fotografía, no la caricatura. Pero al restaurar el buffer, se restaurará de la primera imagen, de esta forma, solo se verán las digitalizaciones, cuando la lente pase sobre las caras de los componentes del grupo. Esto ha sido utilizado por el grupo español Ispania en su demo PUAJJ, creando un efecto realmente vistoso.

LA UNIDAD DEMOVGA

La librería ha crecido, con este artículo. Se han creado dos procedimientos nuevos: Copy64K y ReadRaw. El procedimiento Copy64K transfiere 64000 bytes de una zona de memoria a otra, esto es de especial utilidad cuando se mueven buffers, o imágenes descomprimidas. Sus parámetros de entrada, son el segmento de memoria a copiar, y el segmento de memoria donde se van a copiar 64000 bytes. Es importante destacar que el offset debe ser cero, en ambos casos. El lector es muy libre de realizar las modificaciones oportunas al código adaptándolo a sus necesidades.

El procedimiento ReadRaw lee un fichero gráfico en formato RAW. El formato RAW o vídeo crudo, no lleva ningún tipo de compresión, sencillamente se almacena un byte por cada pixel de pantalla, y al final del fichero, la paleta. Así pues, el tamaño de un fichero RAW en 320x200 será siempre 64000+768. El procedimiento almacena el fichero en el segmento de memoria indicado.

EL BUCLE PRINCIPAL

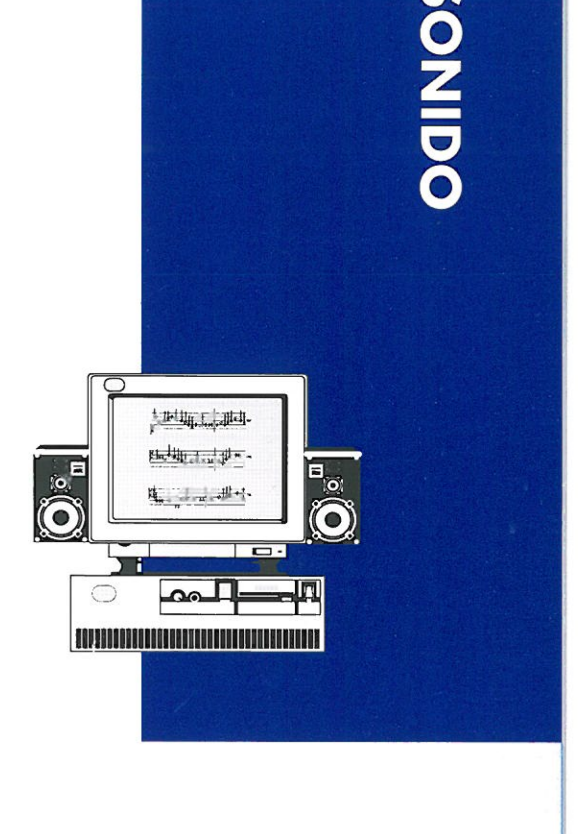
El cuerpo del programa es muy sencillo, se definen unas directrices de movimiento y se sigue el siguiente proceso: se rellena el array de origen con los datos a transformar, se crea la zona destino, con las transformaciones oportunas sobre la zona de origen. Esperamos a que se produzca un retrazo vertical antes de escribir en la memoria de vídeo la zona transformada y se actualiza el movimiento de la lente.

Por ultimo quiero agradecer a Joey su documento en Internet en el que yo aprendí a crear este efecto. Este artículo esta basado en sus explicaciones.

En cuanto a nuestro concurso de Demos e Infografía decir que tenemos la redacción abarrotada, nos espera un duro y largo trabajo de evaluación de todos vuestros trabajos.

EL FORMATO DE FICHERO S3M

Héctor Martínez



Los ficheros S3M se están poniendo más de moda que los propios ficheros MOD que tanto se habían llegado a extender. El motivo principal, es porque están pensados para ser utilizados en PC, y más concretamente con tarjetas Sound Blaster. Tanto es así, que incluso pueden contener tanto instrumentos digitalizados como instrumentos sintetizados FM.. Actualmente ya no se utiliza nada esta capacidad, ya que el sonido FM es muy pobre, y ante la mejora del hardware de los procesadores, se utilizan siempre sonidos digitalizados, que superan con creces al sonido FM de las tarjetas Sound Blaster. Debido a esto, en este artículo sólo se explicará el formato de los instrumentos digitalizados dentro de un fichero S3M, y se dejará de banda el formato de los instrumentos sintetizados.

Otra característica muy importante de este formato, es que permite un máximo de hasta 32 canales de sonido. Ahora no es ninguna novedad, ya que los MOD's ya lo permiten, pero cuando apareció este formato, los MOD's sólo eran de 4 canales, por lo que fue una gran revolución la aparición de este nuevo estándar musical para los ordenadores compatibles PC.

INTRODUCCIÓN A LOS S3M

Un S3M, al igual que el MOD, es un fichero que contiene una composición musical, se compone de las mismas partes de las que se componía el MOD:

Cabecera: contiene datos generales comunes a toda la composición, como el título de la canción, número de instrumentos, volumen, etc.

Instrumentos o samples: Se componen de dos partes, la primera es una descripción del instrumento en cuestión, como la longitud, si tiene o no loop, etc. La segunda no son más que sonidos digitalizados. De la calidad de estas digitalizaciones depende la calidad final que oiremos al interpretar el fichero S3M.

Partitura: es la sucesión de notas musicales, que convenientemente interpretadas resultarán en la melodía. La gran diferencia de ésta respecto al MOD es que está comprimida, es decir, sólo están las notas no vacías. En el MOD, si en un pattern tenía sólo un canal sonando, en el disco estaba escrito ese canal, y además los otros 3 canales en blanco.

El ordenador Amiga tiene cuatro DACs que permiten reproducir cuatro digitalizaciones distintas simultáneamente, cada una a una distinta frecuencia de muestreo. Como el MOD está pensado para estos ordenadores sólo dispone de 4 canales, aunque después, ya en una plataforma PC, se hizo una extensión del MOD a prácticamente cualquier número de canales. Este no es el caso del S3M, que la única limitación que tenían en cuanto a la programación de un player para PC, era la velocidad del procesador, y al ser ésta actualmente tan rápida, permite un máximo de 32 canales.

Al igual que en el MOD, la mezcla de los canales de un fichero S3M, se tiene que hacer mediante el software. Lo referente a como mezclar diferentes canales, cada uno a una frecuencia y a un volumen, no se explicará en este artículo, ya que es exactamente igual que cuando se hizo esto para el formato MOD.

LA CABECERA

La tabla 1 contiene un esquema de la cabecera con cada campo y sus corres-

Diseñados ya sobre ordenadores PC. Es el formato que crea el programa Scream Tracker 3, y que ha sido introducido en las espectaculares demos de Future Crew. Actualmente, es el formato musical preferido por los que se dedican a programar demos, y por los que simplemente pretenden crear su propio módulo musical.

Nombre	Longitud	Descripción
Título	28,00	Título del S3M en ASCIIZ
EOF	1,00	1Ah (End Of File)
FType	1,00	10h
Reservado	2,00	
LongSeq	2,00	Longitud de la secuencia
NumInsts	2,00	Número de instrumentos
NumPatts	2,00	Número de patrones
Flags	2,00	Flags varios
TrackVers	2,00	Versión del tracker
FileVers	2,00	Versión del fichero
Magic	4,00	'SCRM'
VolumenGlobal	1,00	Volumen Global
IniTempo	1,00	Tempo inicial
IniBPM	1,00	BPM inicial
MasterVol	1,00	Volumen total
Reservado2	10,00	
Especial	2,00	
ChnMap	32,00	Channel maps

Tabla 1: Cabecera del S3M.

pondientes longitudes. A continuación se describe con detalle el significado de cada campo:

Titulo: Título de la canción, en formato ASCIIZ, (es decir, ASCII terminado en 0) a no ser que el texto ocupe hasta el último carácter que tiene reservado, en cuyo caso no hay ningún 0.

EOF (1Ah): End Of File, utilizado para poder hacer un type del fichero, y que salga el título del S3M, y no empiece a salir "basura" y pitidos (típicos cuando se visualiza un fichero binario mediante el comando type).

FType (16): Este es el valor que realmente hace que el type finalice.

LongSeq: Longitud de la secuencia de patrones a interpretar.

NumInsts: Número de instrumentos de la composición.

NumPatts: Número de patterns de la composición.

Flags: Indicadores diversos.

TrackVers: Indica la versión del tracker con el que se editó el S3M

FileVers: Versión del fichero.

Magic: En el caso de un S3M tiene que ser siempre SCRM.

VolumenGlobal: Volumen que afecta a todos los canales. Se puede cambiar mediante comandos.

IniTempo: Tempo con el que empezará a interpretarse la partitura.

IniBMP: BMP con el que empezará a interpretarse la partitura.

MasterVol: Volumen master de toda la canción. Se diferencia del volumen global en que no se puede cambiar.

ChnMap: Channel Maps: información relativa a cada canal. El valor, entre 0 y 15, indica el panning de ese canal; un 0 indica que sonará el 100% del columen por el canal izquierdo, y un 15, que lo hará por el derecho. Los valores intermedios ponderan esta distribución.

Un S3M es un fichero que contiene una composición musical

También se utiliza para averiguar el número de canales del S3M, ya que después del último canal, el valor de panning vale más de 128.

Después de lo anterior viene un array de Words (2 bytes por posición) cuya longitud varía según el número de instrumentos. Esta tabla indica para cada instrumento, el offset relativo al principio del fichero de la situación de la cabecera del instrumento. La posición se calcula de esta manera: Valor indicado * 16.

Al igual que el array de posiciones a cabeceras de los instrumentos, viene un array de posiciones a los patrones de la composición. La dirección se calcula de la misma manera.

Formato de la cabecera de cada instrumento

La tabla 2 contiene el formato de la cabecera de los instrumentos con las respectivas longitudes de cada campo.

Tipo: Indica si es FM o digitalizado.

Fichero: Indica el fichero donde está el instrumento, de manera que podemos tener muchos S3M compartiendo los instrumentos, o en cada S3M, incluir los instrumentos dentro.

PosHi, PosLow: Indican mediante un entero de 24 bits el *offset* respecto al principio del fichero donde empieza la digitalización. La manera de calcular el *offset* es: *(PosHi * 65536 + PosLow) * 16.*

Longitud: Longitud de la digitalización en bytes.

LoopStart y LoopEnd: Indican, para instrumentos con loop, el offset de la posición inicial y final del loop. El loop funciona exactamente igual que en los ficheros MOD's, excepto que aquí no hay que hacer manipulaciones adicionales.

Volumen: Es el volumen por defecto del instrumento, es decir, el volumen con el que sonarán todas las notas de ese instrumento a no ser que se especifique lo contrario. Sus valores permitidos van de 0 a 64.

Flags: El bit 0 de este campo nos indica si es un instrumento con o sin loop.

C4Speed: Frecuencia de la nota C4 (DO de la octava 4ª). Hace un efecto similar al Finetune del formato MOD.

Nombre: Nombre o descripción del instrumento, en el mismo formato que el título de la canción. Muchos autores aprovechan estos espacios para poner comentarios como su nombre, fecha de composición, o saludos a amigos, miembros de grupos o incluso dedicatorias amorosas. Esto es debido a que los players muchas veces imprimen por pantalla los nombres de los instrumentos de arriba a abajo, y esto se presta a escribir frases en lugar del nombre del instrumento.

Nombre	Longitud	Descripción
Tipo	1,00	Indica si es FM o digitalizado
Fichero	12,00	Nombre del fichero del sample
PosHi	1,00	Parte alta del offset/16 del sample
PosLow	2,00	Parte baja del offset/16 del sample
Longitud	4,00	Longitud en bytes del sample
LoopStart	4,00	Posición del principio de loop
LoopEnd	4,00	Posición del final de loop
Volumen	1,00	Volumen por defecto para el instrumento
Reservado	1,00	
Pack	1,00	
Flags	1,00	Indica si hay loop o no
C4Spd	4,00	Frecuencia del Do de la octava 4ª
Reservado2	4,00	
Internos	8,00	
Nombre	28,00	Nombre del instrumento
InstId	4,00	Identificador del instrumento

Tabla 2: Cabecera del instrumento.

LOS SAMPLES

Las digitalizaciones pueden estar almacenadas con signo o sin signo. Esto dependerá de la versión del fichero. Si la versión es 1, los samples estarán grabados en formato Amiga (con signo), en versiones posteriores, los samples estarán en formato PC (sin signo). Para ambos casos, las muestras son de 8 bits. Como a la hora de tratar con las muestras, es más cómodo hacerlo con signo, si el formato del fichero es distinto de 1, convertiremos todas las muestras para que sean con signo. La manera de hacer esto es sumar 128 a cada byte. Esta suma se tiene que hacer sobre una variable de tamaño un byte, de manera que 128+128 sea igual a 0 y no a 256.

LA PARTITURA

La partitura de los S3M, está comprimida, sólo ocupa espacio cuando realmente hay algo que hacer en esa posición.

Cada patrón está situado en una posición del fichero indicada por una tabla en la cabecera del fichero. Los primeros dos bytes del patrón, indican la longitud que ocupa ese patrón comprimido (en disco), por lo tanto, los bytes que hay que leer de disco.

Una vez hecho esto, hay que descomprimir el patrón. Esto lo hace la rutina *ConvertPatt*.

Para descomprimir el patrón hay que hacer lo siguiente:

Mirar el primer byte. Este byte tiene dos funciones, indica en que canal estará la nota, y la existencia de cambio de frecuencia, cambio de volumen o efecto. Los 5 bits bajos, indican el número de canal (esto es lo que limita al S3M a 32 canales). Los bits 5, 6 y 7 indican si hay que leer una nueva frecuencia, un nuevo volumen o hay un comando respectivamente. Posteriormente, los siguientes bytes contenidos en el patrón comprimido son en el caso de que lo hayan indicado los bits 5, 6 y 7, y en este orden, la frecuencia, el volumen y el efecto.

Si lo que hay que leer es la frecuencia de una nota, entonces, los dos bytes siguientes indicarán la nota y el instrumento respectivamente. Los cuatro bits más altos indican la octava de la nota (que no debe ser mayor que 11 ni menor que 2) y los cuatro bits bajos indican el número de nota, siendo el 0 correspondiente al DO y el 11 a la nota SI. A partir de aquí hay que calcular la frecuencia tal y como se explicó en el artículo que trató sobre el formato MOD. Luego, esta frecuencia se puede ver modificada según el parámetro *C4Speed* del instrumento al que afecta esa nota. Como el valor normal para la nota C4 es 8363, el divisor que utilizaremos se verá afectado por el factor *8363/C4Speed*.

Si hay que leer el volumen leeremos un byte mas, este byte será el volumen y no debería ser mayor que 64.

Para el caso de un comando, los dos siguientes bytes indican el comando y el parámetro para ese comando respectivamente. El S3M abarca todos los efectos que abarcaba el MOD, y tiene incluso algunos más. El byte que indica el comando, no se corresponde con lo que nos presenta en pantalla el editor de S3M por excelencia (Scream Tracker 3). Este editor, presenta los comandos como caracteres de la A a la Z, por lo tanto, si le sumamos a este byte el valor ASCII de la 'A', tendremos el comando tal y como nos lo muestra el Scream Tracker 3.

Esto se va repitiendo en la misma línea del patrón hasta que encontramos como primer byte un 0. En este caso, hay que pasar a la siguiente línea del patrón.

Los S3M están mucho mas compactados que los MOD

LOS EFECTOS

Procedamos ahora a ver los efectos uno por uno y en orden alfabético. En la tabla 3 están tabulados todos estos comandos:

A: *Set tempo*: El parámetro indica el nuevo tempo con el que se interpreta la partitura.

B: *Position Jump*. Este comando indica un salto en la secuencia de patrones. El parámetro es el número de secuencia de la que hay que obtener el patrón de la siguiente nota, empezando por la primera nota del patrón.

C: *Pattern break*. La siguiente nota se leerá del siguiente patrón. El parámetro indica a partir de qué línea del patrón hay que empezar a leer, se indica en formato BCD:

NotePos := 10(Param SHR 4) + Param AND $F.*

D: *Volume slide*: Si los 4 bits altos del parámetro son unos, entonces el

RAY TRACING (IV) LA REFRACCIÓN

Álvaro Silgado

la refracción es el último fenómeno de la luz que falta por estudiar. gracias a ésta, será posible diseñar las más complejas y reales imágenes que se puedan imaginar.

En el capítulo anterior, se consiguió crear un programa capaz de generar imágenes reales. Si se miran las escenas obtenidas, no se encontrará ningún elemento en ellas que haga desmerecer su realismo, exceptuando un posible hiper-realismo. Es decir, la luz es como debe ser, los objetos aparecen donde deben estar, las reflexiones de la luz son correctas y las sombras generadas se corresponden con la realidad.

Pero el programa todavía tiene ciertas limitaciones. Se pueden definir objetos de cualquier color o patrón de colores, se puede definir el porcentaje de reflexión o coeficiente de reflexión de los objetos, así como su potencial especular (ese halo que aparece en el reflejo de los objetos luminosos). Lo que no puede hacer el programa es permitir que la luz viaje a través de los objetos. O dicho de otra forma, no se pueden obtener objetos transparentes.

La transparencia es un fenómeno que puede resultar relativamente fácil de implementar en el programa, y de echo, lo es. Añadir un objeto como una ventana, que en parte refleje los objetos de la habitación y en parte permita ver lo que hay al otro lado de ella no es muy complicado. Pero una ventana es un objeto muy simple, y la razón es porque está construida un material transparente muy fino y plano. Pero si se pretende incluir un objeto como un vaso con agua, las cosas se complican. En cuanto se incluyan en la escena objetos transparentes de cierto grosor o curvatura, irremediablemente aparecerá el fenómeno de la refracción de la luz.

LA DEFORMANTE REFRACCIÓN

Quien más o quien menos, todos hemos jugado de pequeños a introducir un lápiz dentro de un vaso de agua y ver como éste "se quiebra" al entrar en contacto con el agua, para después salir intacto. Este fenómeno se llama refracción, y se produce siempre que la luz atraviesa un cuerpo y pasa de un medio físico (como el aire) a otro (como el cristal o el agua). El efecto producido es el de cambio de dirección de la luz, y como consecuencia, un aparente desplazamiento de los objetos o de parte de ellos.

Este fenómeno es muy similar en concepto al de la reflexión, aunque un poco más complicado de implementar. Como siempre, lo primero es entender cómo se produce este fenómeno en la realidad, para después reproducirlo lo más fielmente posible en el programa.

La figura 1 muestra un caso típico de refracción de luz. En este esquema, el rayo incidente de luz "I" colisiona en una superficie transparente o semi-transparente, formando un ángulo de incidencia "Ai" con el vector "N", normal a la superficie en el punto de colisión. Como consecuencia del cambio de medio físico, el vector "I" sufre un cambio de dirección, representado por el vector "T" o vector transmitido. Este vector forma un ángulo de refracción o ángulo de transmisión "At" con el vector "-N".

La refracción no siempre afecta mucho a la imagen. Si el objeto que atraviesa es lo suficientemente delgado, apenas es apreciable. En el programa, se ha distinguido entre dos tipos de objetos: abiertos o cerrados. Un objeto cerrado es lo suficientemente ancho como para que se generen rayos en su interior, como por ejemplo, una esfera. Un objeto abierto se supone que no tiene espesor, por lo que permite ver los objetos situados detrás de él, pero sin defor-

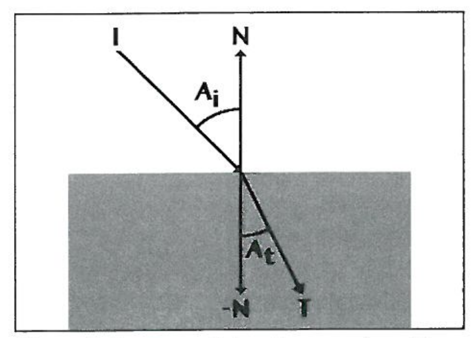


Figura 1. Esquema típico de una refracción.

marlos. Esta característica se ha implementado en la clase objeto.

LAS LEYES DEL JUEGO

Una vez más, es necesario echar un vistazo a los viejos apuntes de física del colegio para refrescar los conceptos. Las leyes físicas que definen la refracción son:

- 1ª Ley o ley de Snell: la relación entre el ángulo de incidencia y el ángulo de transmisión depende tanto del medio físico del que se parte (medio 1) como del medio físico en el que se entra (medio 2), de la siguiente manera:

$$sen\ (At) / sen\ (Ai) = Ni / Nt = Nit$$

siendo

Ni = índice de refracción del medio 1 con respecto al vacío.
Nt = índice de refracción del medio 2 con respecto al vacío.
Nit = índice de refracción del medio 1 con respecto al medio 2.

La figura 2 muestra los índices de refracción de diferentes materiales con respecto al vacío.

- 2ª Ley: El rayo incidente, el vector normal y el rayo refractado están situados en un mismo plano, es decir, el vector refractado se puede expresar como una combinación lineal de los vectores incidente y normal:

$$T = a * I + b * N$$

ÁLGEBRA Y TRIGONOMETRÍA

A continuación se va a exponer un desarrollo matemático de cierta complejidad. El lector que no esté interesado en este tipo de desarrollos puede perfectamente saltarse este capítulo, aunque no es aconsejable de cara a un mejor entendimiento del programa. Todos los pasos están perfectamente comentados y es de esperar que sean entendidos completamente.

De lo que se trata es de calcular el rayo refractado T. Para ello, hay que tener en cuenta que todos los vectores están normalizados (su longitud o normal vale 1). De esta forma, se puede ver claramente que

$$cos\ (Ai) = Ci = N | -I$$
$$cos\ (At) = Ct = -N | T$$

Hay que recordar que el símbolo "|" representa el producto escalar de dos vectores, cuyo valor es el coseno del ángulo que forman dividido por el producto de sus normales. Como ambos vectores están normalizados, el resultado obtenido es el coseno del ángulo que forman entre sí.

La ley de Snell anteriormente comentada hace referencia a los senos de los ángulos. Para cambiarla de forma que trabaje con los cosenos es necesario recordar un principio básico de la trigonometría:

$$(sen\ A)^2 + (cos\ A)^2 = 1$$

Despejando el seno,

$$(sen\ A)^2 = 1 - (cos\ A)^2$$

De esta forma, sustituyendo en la ley de Snell elevada al cuadrado, se obtiene

$$(1 - cos\ (At)^2) / (1 - cos\ (Ai)^2) = Nit^2$$

O, lo que es lo mismo,

$$(1 - Ci^2) * Nit^2 = 1 - Ct^2$$

Dejando a un lado de la ecuación Ct^2 y sustituyéndolo por la expresión obtenida anteriormente, se obtiene

$$(1 - Ci^2) * Nit^2 - 1 = -(-N | T)^2$$

Ahora, es necesario sustituir T en la expresión anterior, de tal forma que se aplique la segunda ley de la refracción:

$$(1 - Ci^2) * Nit^2 - 1 = -(-N | (a * I + b * N))^2$$

Desarrollando el producto escalar,

$$(1 - Ci^2) * Nit^2 - 1 = -(a * (-N | I) + b * (-N | N))^2$$

Hay que recordar que el producto escalar de un vector normalizado consigo mismo vale 1, ya que forma un ángulo de 0 grados y el coseno de 0 es 1. Por lo tanto, $-N | N = -1$. Sustituyendo esto en la ecuación,

$$(1 - Ci^2) * Nit^2 - 1 = -(a * Ci - b)^2$$

Ya hay una ecuación. El problema es que hay dos incógnitas: "a" y "b". Por lo tanto, es necesario que exista una segunda ecuación. Y el truco para obtenerla es tener en cuenta que el vector resultante "T" tiene que estar normalizado. Por lo tanto,

$$T | T = 1$$

Si se sustituye la expresión de "T" en la ecuación anterior,

$$(a * I + b * N) | (a * I + b * N) = 1$$

Desarrollando de nuevo el producto escalar:

$$a^2 * (I | I) + 2 * a * b * (I | N) + b^2 * (N | N) = 1$$

Como antes, se sustituyen los productos escalares iguales a 1 y se obtiene la segunda ecuación. El objetivo ahora es resolver este sistema de ecuaciones de segundo grado:

$$a^2 - 2 * a * b * Ci + b^2 = 1$$
$$(1 - Ci^2) * Nit^2 - 1 = -(a * Ci - b)^2$$

Para poder resolver este sistema de ecuaciones, es necesario desarrollarlas lo más posible:

El vector incidente, el refractado y el normal están en un mismo plano

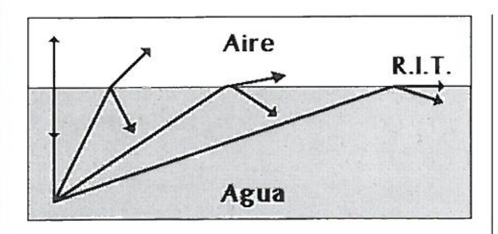


Figura 3. Índices de refracción más importantes.

$-2 * a * b * Ci + b \wedge 2 = 1 - a \wedge 2$
$Nit \wedge 2 - Ci \wedge 2 * Nit \wedge 2 - 1 + a \wedge 2 * Ci \wedge 2 - 2 * a * b * Ci + b \wedge 2 = 0$

Ha habido suerte, el término de la izquierda de la primera ecuación aparece en la segunda. Sustituyendo de una ecuación en la otra:

$Nit \wedge 2 - Ci \wedge 2 * Nit \wedge 2 - 1 + a \wedge 2 * Ci \wedge 2 + 1 - a \wedge 2 = 0$

Ahora, se agrupan los términos comunes a *"a ^ 2"* y a *"Nit ^ 2"*, quedando:

$Nit \wedge 2 * (1 - Ci \wedge 2) - a \wedge 2 * (1 - Ci \wedge 2) = 0$

Seguimos de suerte. Hay un término común que puede pasar al otro lado de la ecuación dividiendo. Como en el otro lado hay un 0, se elimina:

$Nit \wedge 2 - a \wedge 2 = 0$

Por lo tanto, los posibles valores que puede tomar "a" son:

$a1 = Nit$
$a2 = - Nit$

Con estos valores, y sustituyendo en una de las ecuaciones, se obtienen los valores de b, que son:

$b = a * Ci \pm sqrt (1 + Nit \wedge 2 * (Ci \wedge 2 - 1))$

Como "a" puede tomar 2 valores y "b" también, se obtienen cuatro posibles resultados como consecuencia de las combinaciones de valores. De las cuatro combinaciones, la correcta es:

$a = - Nit$
$b = Nit * Ci - sqrt (1 + Nit \wedge 2 * (Ci \wedge 2 - 1))$

Las otras combinaciones se corresponden con los vectores simétricos respecto al vector normal y a la superficie del objeto. Si se sustituyen estos valores en la expresión de "T", se obtiene la tan buscada solución:

$T = (Nit * Ci - sqrt (1 + Nit \wedge 2 * (Ci \wedge 2 - 1))) * N - Nit * I$

REFLEXIÓN INTERNA TOTAL

Aunque el título pueda desconcertar un poco, no se trata de un estado avanzado de meditación trascendental. Es un fenómeno que se produce cuando un rayo incide en una superficie formando un ángulo demasiado grande con respecto al vector normal, no permitiendo que el rayo pueda atravesar la superficie y pasar al otro medio físico. En estos casos el vector refractado no se genera y, por lo tanto, no es necesario calcularlo. La figura 3 muestra como a medida que se va abriendo el ángulo de incidencia, el de refracción se va cerrando hasta que llega un punto en el que se produce el fenómeno de reflexión interna total (R.I.T.).

Para detectar cuándo se produce la R.I.T., basta con comprobar en la fórmula anterior cuándo el contenido de la raíz cuadrada es negativo. Es estos casos, se estará produciendo este fenómeno y no será necesario continuar con los cálculos de ese rayo.

MIRANDO A TRAVÉS

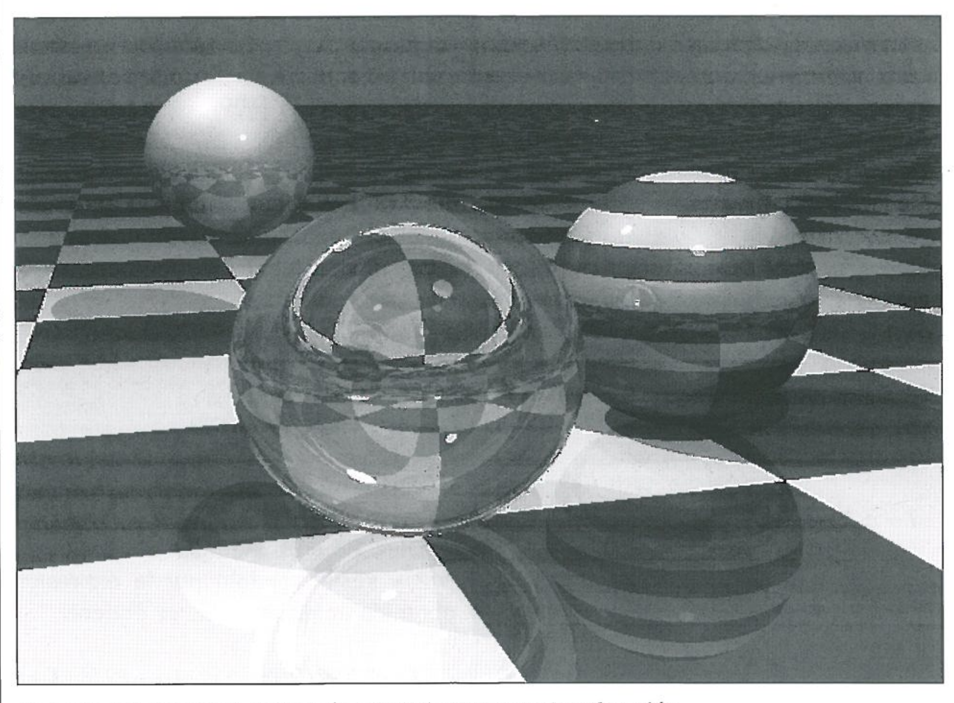
Todo esto y todo lo anterior es lo que se realiza en la función rayo.refractado, que devuelve 1 en caso de que se pueda calcular el rayo refractado y 0 en otro caso.

Como ocurre con la reflexión, también va a existir un coeficiente de refracción o coeficiente de transmisión llamado kt. Este coeficiente mide el porcentaje de luz que refracta el objeto. Se ha modificado la clase material para que contenga este campo nuevo. De esta forma tenemos tres tipos de luz que van a contribuir al resultado final del objeto, con sus coeficientes correspondientes:

kr = porcentaje de luz reflejada.
kt = porcentaje de luz refractada.
1 - kr - kt = color del objeto.

De esta forma, si kr + kt = 1, no será necesario calcular el color del objeto como consecuencia de la iluminación. Este es el caso de objetos de cristal incoloro.

Un cambio importante con respecto a la versión anterior del programa es que ahora el rayo puede estar viajando por un medio diferente al aire. Por lo tanto, esta información debe acompañar al rayo en todo momento. Se ha modificado la clase rayo para cubrir esta necesidad. Esta versión es muy



La imagen de siempre, pero más impresionante con la refracción.

MEDIO FÍSICO	ÍNDICE DE REFRACCIÓN
Agua	1.33
Alcohol	1.36
Aire	1.0003
Cuarzo fundido	1.46
Cristal	1.52
Cristal denso	1.66

Figura 2. Reflexión Interna Total.

simplificada, y sólo se permite que viaje por aire o por cristal. En el módulo rt.hpp se han definido los índices de refracción de estos dos medios físicos.

Hay otro cambio que debe ser aclarado. Si se contempla la figura 3, puede observarse que cada rayo que parte del interior de un objeto puede generar un rayo refractado que sale del objeto y un rayo reflejado que permanece en su interior. Para que este rayo reflejado sea calculado correctamente, es necesario cambiar de signo el vector normal siempre que el rayo esté en el interior de un objeto. Esto se hace con la siguiente sentencia:

```
if (r.medio != AIRE)
  normal = - normal;
```

De esta forma, el algoritmo principal de trazado de un rayo queda de la siguiente manera:

- Trazar el rayo que pasa por el píxel a estudiar.
- Comprobar si hace intersección con algún objeto y, en caso afirmativo, quedarse con el que lo hace más cerca del origen del rayo.
- Si kt + kr < 1, estudiar la iluminación en el punto de intersección para obtener la luz difundida por el objeto.
- Si el rayo es interior, invertir el vector normal.
- Si procede, calcular el rayo reflejado en el punto de intersección y trazar este nuevo rayo de forma recursiva, promediando el color del objeto con el color obtenido con el rayo reflejado.
- Si procede, calcular el rayo refractado en el punto de intersección y proceder de igual manera que con el rayo reflejado. El color resultante será:

*kr * luz_reflejada + kt * luz_refractada + (1 - kr - kt) * luz_difusa*

El índice de refracción varía según los materiales

TRANSPARENCIAS LIMITADAS

Al igual que en la reflexión estudiada en el capítulo anterior, no siempre se va a evaluar la refracción en un objeto. Para que un rayo no esté infinitamente viajando a través de los objetos de la escena, se le impone un límite en cuanto al número de cambios de medio que puede hacer. También se limita el número de rayos generados acumulando los coeficientes de refracción de los objetos que va atravesando, hasta que se obtenga un valor muy pequeño. En ese momento, no merece la pena continuar con ese rayo.

Por lo tanto, además de pasar las variables de limitación de la reflexión a la función trazar_rayo, también se le pasa las de limitación de la refracción. El funcionamiento de éstas es idéntico a las explicadas en el capítulo anterior. Con un máximo de 2 reflexiones y 3 refracciones por rayo se consiguen imágenes tan espectaculares como la de la figura 4.

La refracción aporta un gran realismo a la escena

SOMBRAS DE TRANSPARENCIA

Las transparencias generan otras alteraciones además de las estudiadas: las sombras ya no van a ser negras. La explicación de esto es que la luz ahora puede viajar a través de ciertos objetos, por lo que sus sombras no son del todo oscuras.

En la realidad, la sombra que se genera no es uniforme, sino que depende de la forma del objeto al que pertenece. Si se contempla, por ejemplo, una canica de cristal iluminada con una lámpara, la sombra en el centro es más brillante, mientras que en los extremos es prácticamente negra. Reproducir este fenómeno es bastante costoso en tiempo, ya que requiere muchas operaciones matemáticas. Normalmente, se suele hacer una aproximación lo suficientemente buena como para engañar al ojo.

Esta aproximación consiste en ir multiplicando el valor de la luz que proviene de un objeto luminoso por los coeficientes de refracción de los diferentes objetos que va atravesando. De esta forma, se va oscureciendo poco a poco, o completamente si topa con un objeto opaco. Si no topa con ningún objeto, no se produce sombra. Si sólo topa con objetos transparentes, la sombra es tenue. Si topa con un objeto opaco, la sombra es oscura.

Para implementar esto, es necesario retocar un poco la función iluminar. Al calcular la luz que recibe un objeto, en lugar de parar el bucle al encontrar un objeto que haga sombra, se para el bucle cuando la luz es suficientemente pequeña. Por lo demás se procede de igual forma que antes.

SE ACABÓ LA FÍSICA

Con este capítulo se termina de estudiar todos los procesos físicos que influyen en un programa de Ray Tracing. Lo que se va a estudiar a partir de ahora son temas relacionados con el material de los objetos, como texturas; nuevos objetos, como conos, cilindros, toros, etc.; objetos especiales, como curvas cuádricas, fractales, etc.; así como otros temas relacionados con este mundo tan fantástico.

BIBLIOGRAFÍA

- "An Introduction to Ray Tracing". Editado por Andrew S. Glassner. Editorial Academic Press.
- "Graphics Gems". Editado por Andrew S. Glassner. Editorial Academic Press.
- "Fundamentals of Three-Dimensional Computer Graphics". Alan Watt. Editorial Addison-Wesley.
- "Computer Graphics. Principles and practice". Foley, van Dam, Feiner y Hughes. Editorial Addison-Wesley.

BASES DE DATOS: DB2

José María Peco

Siguiendo con la línea iniciada hace dos meses, este mes le toca el turno al sistema gestor Relacional, y para ello se ha elegido el que cuenta con mayor número de licencias instaladas en los grandes centros de procesos de datos de España, el DB2 de IBM.

Este sistema de gestión es un sistema basado en el modelo relacional. Esto quiere decir que las relaciones entre los distintos ficheros (o tablas, usando la terminología relacional) se establece mediante los propios valores de los campos, no mediante apuntadores como se hace en los modelos jerárquicos, en red, o basados en listas invertidas, vistos anteriormente.

Para poder llevar a cabo su trabajo, este gestor mantiene un catálogo con la información que necesita. Este catálogo esta formado por las 30 tablas que se enumeran en el cuadro de la figura 1 junto con una breve descripción. La figura 2 muestra un fragmento del modelo de datos con las relaciones existentes entre las distintas tablas que componen dicho catálogo.

TERMINOLOGÍA

Una de las particularidades de este modelo es el uso de una terminología especial, por eso en este apartado se enumeran aquellos términos mas comunes del entorno relacional.

Tabla o relación: Es un conjunto o lista de elementos homogéneos que puede representarse mediante una matriz de dos dimensiones, en la que cada una de las columnas establece los atributos o características definibles a cada uno de los elementos de la tabla, y por lo tanto cada fila contiene todas las características de una ocurrencia.

Grado de una tabla es el número de atributos definidos en una tabla.

Dominio de un atributo: Conjunto de valores que podrán usarse como contenido del atributo especificado.

Atributo o columna: cada campo de un registro,

Fila o tupla : cada uno de los registros de una tabla.

Entidad: elemento del modelo de datos que se implementa físicamente mediante una tabla en la base de datos.

CLAVES

Un concepto importante, y básico para este gestor, es que cada una de las ocurrencias (registro) de una tabla debe estar perfectamente identificada mediante al menos un atributo (campo) único, pues a la hora de recuperar o actualizar cada ocurrencia, solo podremos referenciarla con ese valor.

Las claves, como se puede intuir, juegan un papel muy importante en este modelo, ya que van a facilitar la recuperación de los datos, debiéndose diferenciar los siguientes tipos de clave:

Clave candidata: Atributo o conjunto de atributos que permiten identificar cada ocurrencia de una tabla de forma unívoca. Ejemplo: Cod_empleado y DNI_empleado.

Clave principal: La clave elegida como tal entre las distintas claves candidatas. Se suele representar por PK (primary Key). Ejemplo: COD_empleado

Clave Externa: Son aquellos atributos de una tabla que son clave Primaria en otra (FK:Foreign Key). De esta forma es como se establecen las relaciones, migrando la clave primaria de una tabla como claves externas de todas aquellas con los que se establezca la relación. Ejemplo: Cod_provincia.

ÍNDICES

A la hora de implementar físicamente el modelo de datos relacional, y con objeto de aumentar la eficiencia con la

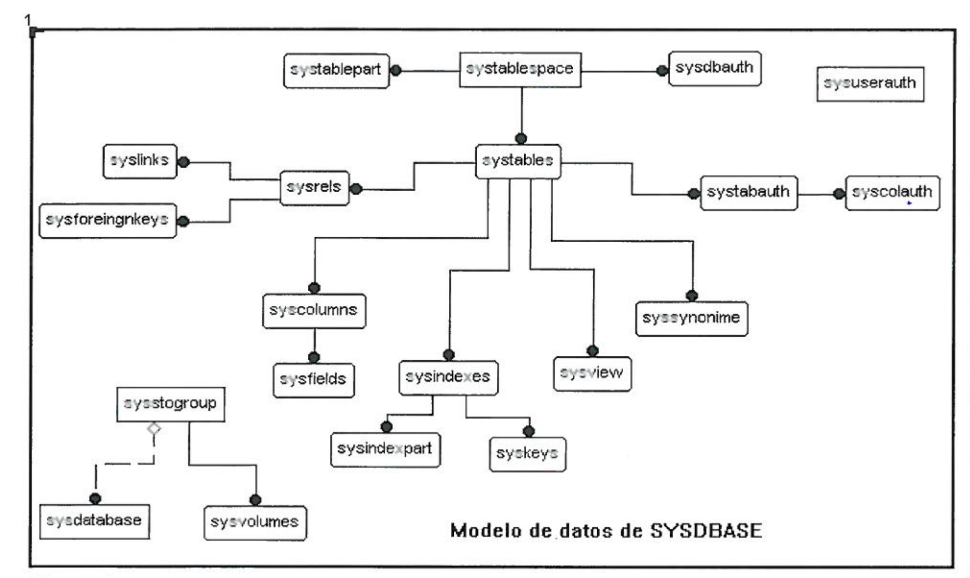


Figura 1.

que DB2 debe acceder a los datos, se pueden definir índices físicos, los cuales son como cualquier índice de cualquier sistema de ficheros, estando cada entrada compuesta por el valor de la clave simbólica y un identificador de registro o RID, siendo este RID una numeración interna que identifica de modo único cada fila para DB2.

Físicamente el RID esta compuesto de los siguientes elementos

- Número de página: (3 Bytes) siendo la página, el espacio físico de la tabla donde se almacena la fila (registro).
- Identificador (ID) (1 Byte) el numero de la entrada de la Tabla de localización de registros (TLR) de la página referenciada en los tres primeros Bytes.

El número de índices que se pueden definir sobre una tabla depende principalmente del modo en el que se van a procesar las filas en ella contenidas, ya que los Índices reducen las operaciones de I/O en los procesos de consulta, pero los aumentan cada vez que se actualiza una columna que forma parte del índice, o cuando se borran o añaden filas.

Dentro de este apartado, se debe resaltar el hecho de que en DB2 no es necesario definir índices, ya que en este caso, el gestor, para acceder a los datos de una tabla, realiza un barrido de las páginas del espacio definido para esa tabla. Por regla general, las tablas que ocupen menos de 4 páginas (1 página = 4k) no suelen tener índices definidos, ya que, aunque los tengan, DB2 al decidir el medio que usará para recuperar la información, optará por el barrido físico de la tabla. En cambio, las de tamaño superior, suelen tener asociado un índice, que se denomina cluster, y que se suele corresponder con la clave primaria de la tabla.

Cuando existe un índice cluster, DB2 intentará almacenar las filas en las páginas de la tabla, en la misma secuencia que se encuentren las entradas en el índice cluster, es decir, las agrupará físicamente en base al valor del índice cluster.

Fila
Zorba | Segovia | 123456
Indice
Cliente RID
Garcia 068-3
Perez 124-5
Zorba
espacio de tablas
Pagina
Y
TLR Y X
147 148 149
1 4 8 2
Pagina Entrada
TLR Table Location Record

Figura 2.

ARQUITECTURA

Toda la información gestionada por DB2, el propio gestor la agrupa en subsistemas, entendiendo por tal, un completo, independiente y autónomo entorno DB2, lo cual quiere decir que:

- Tiene su propio catálogo
- Puede instalarse en la misma CPU que otro subsistema, o en CPUs separadas.
- Puede tener un interface con otros subsistemas vía DDF (Distributed Data Facility)

La figura 3 muestra de forma esquemática todos los elementos de la arquitectura de la Base de datos. Físicamente las distintas ocurrencias de una tabla se localizan en las páginas que tiene asignadas la tabla a la que pertenece dentro de su *TABLESPACE* . Cada una de estas páginas dispone en las últimas posiciones de la misma, de una tabla (la TLR) en la que cada entrada contiene los desplazamientos relativos a principio de página de cada una de las ocurrencias almacenadas en dicha página.

Por otra parte, se ha querido mostrar en la misma figura la composición de un índice, y cómo se puede localizar mediante el RID un registro.

EL LENGUAJE RELACIONAL:

Este es el segundo de los componentes principales de un modelo relacional, ya que permite la recuperación/actualización de los datos mediante el uso de un lenguaje preciso y estructurado, el SQL (Structured Query Language), el cual además se ajusta a los estándares ISO y ANSI.

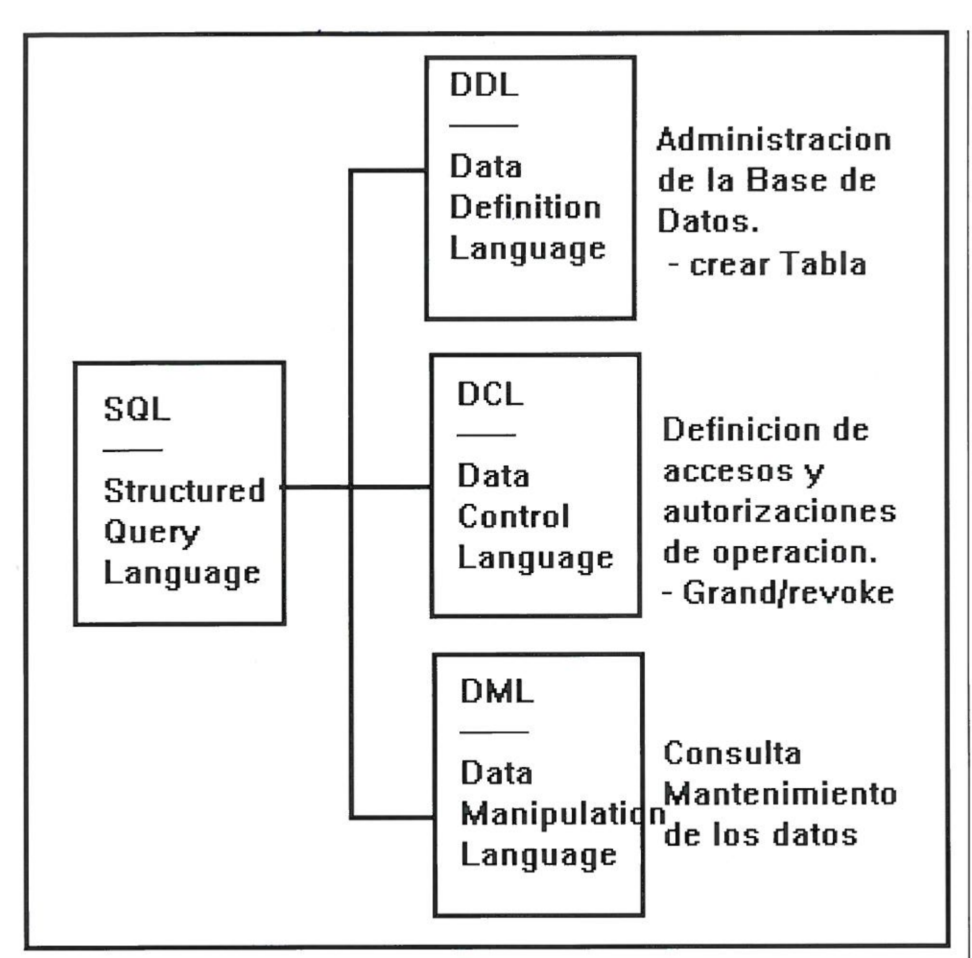


Figura 3.

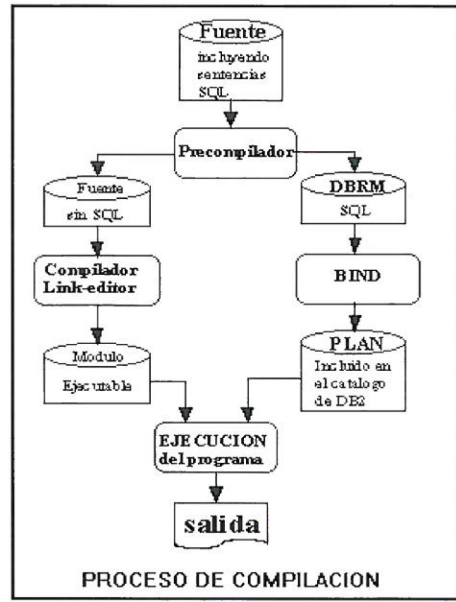


Figura 5.

La figura 4 muestra los subconjuntos en los que se pueden agrupar los distintos comandos de este lenguaje, así como el uso al que se destinan, siendo, desde el punto de vista del programador, el SQL/DM el que mas importancia tiene, ya que es el que tendrá que manejar mas frecuentemente para acceder a los datos.

El criterio usado para este acceso es:

- QUÉ columnas se quiere recuperar
- DESDE qué tabla o vista
- DÓNDE va a colocar la información leída
- en qué ORDEN va a proporcionar los datos.

Como se puede apreciar en el ejemplo de la figura 4, básicamente, la idea que predomina en SQL, es la de especificar

```
SELECT cod_cliente, nom_cliente, tel_cliente
FROM   tab_cliente,his_cliente
WHERE apel_cliente="perez"
```

Ejemplo :

Recuperar los datos de código, nombre y telefono de la tabla de CLIENTES y de la tabal HISTORICO de clientes, correspondientes al cliente que tiene por primer apellido PEREZ .

Figura 4.

qué el lo que se quiere, y no especificar cómo debe hacerlo el gestor.

Esto obliga a introducir algunos conceptos nuevos en el proceso de compilación de los programas, quedando éste como se muestra en la figura 6 que se acompaña.

Así, un programa escrito en cualquier lenguaje convencional de programación que quiera usar datos almacenados en DB2, deberá incluir sentencias SQL enmarcándolas entre las palabras reservadas *EXEC* y *END-EXEC* con las que poder decir al gestor, qué información es la que desea recuperar. El primer paso por tanto, para compilar el programa, debe consistir en eliminar esas sentencias que no son del lenguaje en el que se escribe el programa, y que por lo tanto, no entenderá el compilador. En este proceso de precompilación pasan a ser comentarios las sentencias SQL, siendo sustituidas por sentencias *CALL* que invocan a determinados módulos. Así mismo, en este primer paso, se extraen a otro fichero todas las sentencias SQL, para su posterior tratamiento, constituyendo los módulos denominados DBRM.

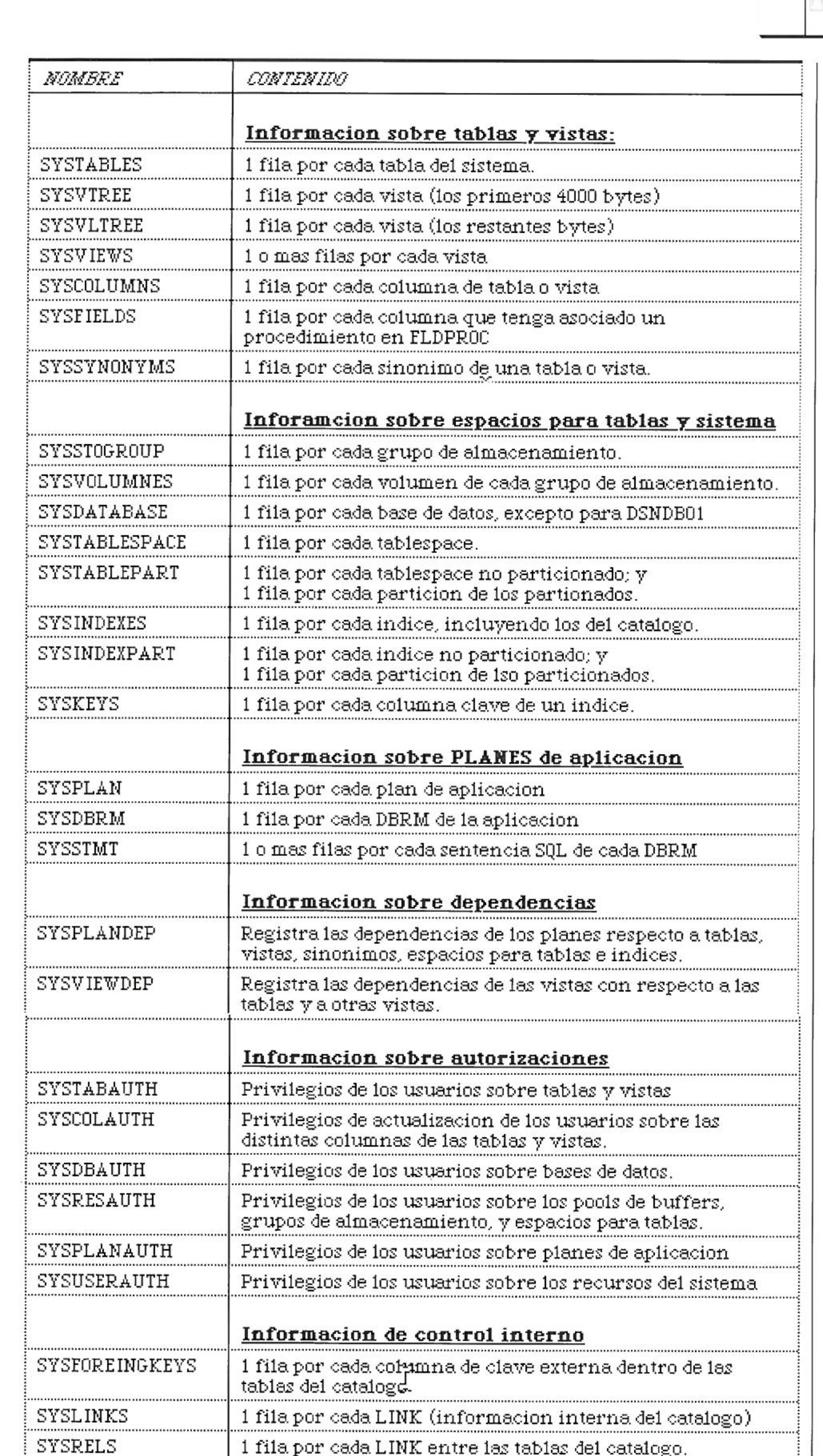
En el siguiente paso, se compila el programa principal, pues ya todas las sentencias son del lenguaje conocido por el compilador, obteniéndose un modulo objeto, para convertirse posteriormente después de pasar el proceso del montador, en modulo ejecutable.

Por otra parte, a los módulos DBRM que se obtuvieron en el proceso de precompilación, se les debe aplicar el proceso de *BIND* a fin de establecer la estrategia de acceso a los datos que seguirá DB2 cuando sea invocado. Esta estrategia, o elección del camino a seguir para la ejecución de una sentencia SQL, se toma en base a las estadísticas que lleva el propio gestor, tales como número de páginas que ocupa la tabla, índices definidos, etc. El resultado de esta estrategia es almacenada por DB2 en una de sus tablas de gestión, y es lo que constituye un PLAN.

De esta forma, es la combinación Modulo-ejecutable/Plan lo que permite la ejecución de un programa que trabaje contra tablas DB2.

INTEGRIDAD REFERENCIAL

Con objeto de asegurar la integridad referencial, cuando se selecciona una fila para modificarla, se bloquea no sólo el registro seleccionado, sino toda la página en la que se encuentra, pues es a este nivel en el que se encuentran unos indicadores dispuestos para tal efecto, como ya se expondrá en próximos artículos.



NOMBRE	CONTENIDO
	Informacion sobre tablas y vistas:
SYSTABLES	1 fila por cada tabla del sistema.
SYSVTREE	1 fila por cada vista (los primeros 4000 bytes)
SYSVLTREE	1 fila por cada vista (los restantes bytes)
SYSVIEWS	1 o mas filas por cada vista
SYSCOLUMNS	1 fila por cada columna de tabla o vista
SYSFIELDS	1 fila por cada columna que tenga asociado un procedimiento en FLDPROC
SYSSYNONYMS	1 fila por cada sinonimo de una tabla o vista.
	Inforamcion sobre espacios para tablas y sistema
SYSSTOGROUP	1 fila por cada grupo de almacenamiento.
SYSVOLUMNES	1 fila por cada volumen de cada grupo de almacenamiento.
SYSDATABASE	1 fila por cada base de datos, excepto para DSNDB01
SYSTABLESPACE	1 fila por cada tablespace.
SYSTABLEPART	1 fila por cada tablespace no particionado; y 1 fila por cada particion de los partionados.
SYSINDEXES	1 fila por cada indice, incluyendo los del catalogo.
SYSINDEXPART	1 fila por cada indice no particionado; y 1 fila por cada particion de lso particionados.
SYSKEYS	1 fila por cada columna clave de un indice.
	Informacion sobre PLANES de aplicacion
SYSPLAN	1 fila por cada plan de aplicacion
SYSDBRM	1 fila por cada DBRM de la aplicacion
SYSSTMT	1 o mas filas por cada sentencia SQL de cada DBRM
	Informacion sobre dependencias
SYSPLANDEP	Registra las dependencias de los planes respecto a tablas, vistas, sinonimos, espacios para tablas e indices.
SYSVIEWDEP	Registra las dependencias de las vistas con respecto a las tablas y a otras vistas.
	Informacion sobre autorizaciones
SYSTABAUTH	Privilegios de los usuarios sobre tablas y vistas
SYSCOLAUTH	Privilegios de actualizacion de los usuarios sobre las distintas columnas de las tablas y vistas.
SYSDBAUTH	Privilegios de los usuarios sobre bases de datos.
SYSRESAUTH	Privilegios de los usuarios sobre los pools de buffers, grupos de almacenamiento, y espacios para tablas.
SYSPLANAUTH	Privilegios de los usuarios sobre planes de aplicacion
SYSUSERAUTH	Privilegios de los usuarios sobre los recursos del sistema
	Informacion de control interno
SYSFOREINGKEYS	1 fila por cada columna de clave externa dentro de las tablas del catalogo.
SYSLINKS	1 fila por cada LINK (informacion interna del catalogo)
SYSRELS	1 fila por cada LINK entre las tablas del catalogo.

Hay que tener en cuenta que no es suficiente con identificar las claves primarias de una entidad con la externa de otra, para asegurar la integridad, sino que hay un conjunto de reglas que se tienen que aplicar a cada relación cuando se insertan o actualizan filas con claves externas o cuando se borran claves primarias. Así, cuando se intenta borrar una fila de una clave primaria, las reglas de integridad referencial definen lo que debe ocurrir en el resto de tablas que contienen como clave externa el valor borrado, debiéndose aplicar una de las siguiente reglas:

- Borrado restringido: No se permite borrar la fila de la clave primaria.
- Borrado con neutralización: Se ponen a nulo todos los valores de la clave externa que son iguales a la clave primaria que se quiere borrar.
- Borrado en cascada: Todas las filas de la clave externa con un valor igual a la clave primaria de la fila que se borra, se borran también.

Reglas similares deben aplicarse en inserción, pero en este caso, es el valor especificado como clave externa el que debe existir en la columna de la clave primaria de la tabla padre. Y, en actualización, el nuevo valor especificado en una clave externa tiene que ser igual a un valor existente en la tabla padre, o tener valor nulo.

SEGURIDAD EN DB2

Por último, se quiere resaltar otro tema muy importante en DB2. Es el tema de las autorizaciones. IBM tomó desde el principio la decisión de que cualquier objeto creado por DB2 debería ser controlado por DB2. Mucho se podría hablar sobre este tema, pero con objeto de no dejarle en el tintero simplemente se enuncian las tres fuentes de autorización que hay en DB2:

- Creando Objetos DB2: Como creador de un objeto, se tienen ciertos privilegios sobre dicho objeto, sin necesidad de haber sido autorizado.
- Autorización administrativa: Son grupos de privilegios que se unen a entornos de trabajo de DB2.
- El comando *GRANT*: esta sentencia SQL/DC se usa para asignar privilegios.

En fin, como ya se ha dicho anteriormente, cada uno de los puntos enunciados en estos tres artículos, pueden ser la introducción de un tema monográfico mucho mas amplio, pero la idea que movió al autor de estos artículos fue la de proporcionar al desarrollador una visión general de los SGDB desde la perspectiva de sistemas, pues este conocimiento facilita en muchas ocasiones el desarrollo de programas.

INSTALACIONES A MEDIDA

David Aparicio

La distribución Slackware, suministrada normalmente en esta revista, nos permite una instalación genérica. Sin embargo, es posible ajustar los discos que se suministran, para automatizar dicha instalación.

El sistema de distribución Linux más extendido hoy día es el denominado Slackware. Como todas las instalaciones complejas, que manejan decenas de megas de datos, se requiere una cierta organización en la configuración. Esto aporta mayor sencillez para los nuevos usuarios, y también para los suministradores, que pueden producir distribuciones diferentes con cambios mínimos.

Este mes vamos a ver los ficheros de configuración que nos permitirán generar unos discos de instalación personalizados. Si instalamos Linux sólo para nosotros mismos, este artículo puede ayudarnos a arreglar pequeños fallos en distribuciones defectuosas. Con grandes volúmenes de información, la posibilidad de que el autor de una distribución cometa algún fallo es relativamente normal. Sin embargo, con un mínimo de conocimientos, podremos solventar fácilmente estas erratas.

Por otro lado, si solemos instalar Linux en varias máquinas, y queremos evitar operaciones repetitivas y tediosas, una preconfiguración de la distribución que empleemos puede ser la solución que estamos buscando.

UNA TOMA DE CONTACTO

Si usted ya ha instalado alguna vez Linux-Slackware, lo que comentaremos a continuación le debería resultar familiar. Slackware divide el sistema por temáticas, en grupos de discos conocidos como series, los cuales reciben una denominación sencilla, de una o varias letras. Por ejemplo, la serie A contiene el sistema básico, AP son aplicaciones de texto, X es la serie de ventanas, etcétera. Cada serie se divide en varios fragmentos, que agrupan ficheros comprimidos con un espacio menor de 1440 Kbytes. De este modo, cada fragmento de una serie determinada puede caber en discos flexibles de 3 pulgadas y media, y posibilita el acceso de Slackware a todos los usuarios que no disponen de lector de CD o conexión a una red. Para los que sí disponen de alguno de los medios mencionados, cada disco se encuentra contenido en un subdirectorio independiente, que lleva el nombre del propio disco (a1, a2, ap1 ...).

En caso de instalación con disco flexible, éstos se encuentran formateados en MS-DOS para poder ser consultados sin tener acceso a un ordenador con Linux. Adicionalmente, encontraremos dos discos adicionales.:

- El primero de ellos es el de arranque, *boot*, y contiene una imagen del núcleo Linux que se utilizará sólo mientras dure la instalación. Una vez que el sistema está completamente instalado, el núcleo que se utilizará será alguno de los que se encuentre en la serie A o Q, o bien uno recompilado por nosotros mismos.
- El segundo, el disco principal de instalación, *root*, está en formato "minix", y contiene los programas (*setup, fdisk, mkfs, mkswap, mount* ...) que nos permitirán preparar el PC para iniciar Linux en él.

En este artículo nos centraremos en los discos que componen las propias series, y dejaremos aparte estos dos últimos.

FICHEROS DE CONFIGURACIÓN: "TAGFILES"

En este apartado, vamos a dar un breve recorrido por los diferentes tipos de ficheros contenidos en los discos de

Slackware. Los ficheros cuya extensión es tgz, son los que contienen los programas a instalar en nuestro disco duro. Debido a que los componentes de las series Slackware están en formato DOS, todos los nombres se ajustan a la longitud de "8 más 3" caracteres a que nos limita dicho formato.

Además de éstos, encontraremos los siguientes:

• **diskXX:** Cada disco de cada serie tiene un fichero que lo identifica de forma única, y cuya existencia es obligatoria para ser reconocido por el programa de instalación, Por ejemplo, el tercer disco de la serie N contendrá el fichero denominado "diskn3". El contenido del mismo es una lista de los archivos tgz del disco, más una descripción de los mismos.
• **install.end:** El último disco de cada serie viene marcado por la existencia de este fichero. No necesita contener información, puesto que sólo sirve para marcar el final de cada serie.
• **tagfile:** Este fichero se encuentra sólo en el primer disco de cada serie, y describe el grado de importancia de cada archivo para la instalación. Se utiliza cuando elegimos "default tagfiles" al realizar la instalación (figura 1).
• **tagfile.org:** Es una copia original del fichero anterior, por si cometemos un error al editarlo y deseamos recuperar los datos de partida. Nunca se utiliza en una instalación.
• **tagfile.pat:** El fichero de configuración "recomendado" por Patrick Volkerding, el autor de las distribuciones Slackware. Representa una alternativa a la instalación por defecto, y tenemos oportunidad de seleccionarlos si elegimos "custom tagfiles" en el setup. Tanto este fichero como el anterior se encuentran en el primer disco de cada serie.
• **maketag:** Es un ejecutable que permite construir un fichero de configuración (tagfile) a medida, de forma automatizada. Se invoca con la opción "MAKE TAGS" del menú principal (figura 2) en el programa setup.

Una vez conocidos los ficheros, pasaremos a analizar el formato de cada uno de ellos.

El fichero "diskXX" tiene un contenido similar a:

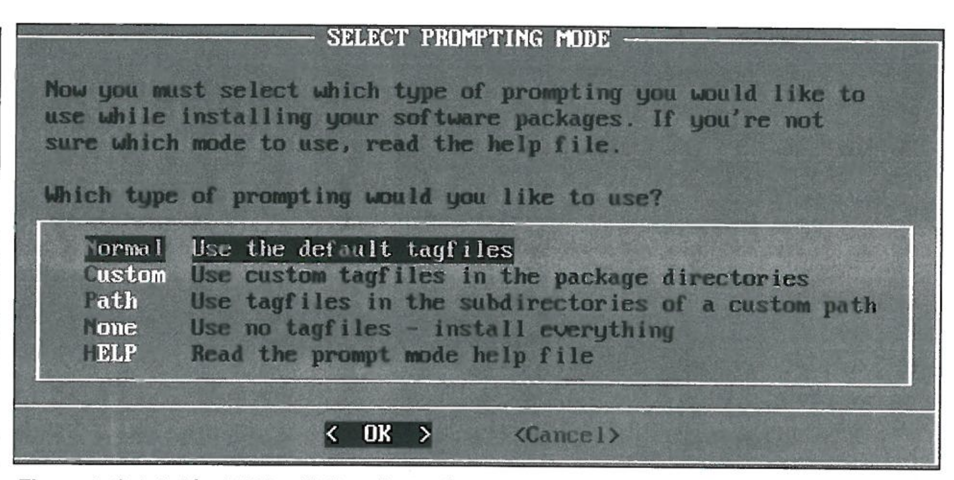


Figura 1: Selección del "tagfile" en "setup".

CONTENTS: archivo1 archivo2 archivo3
archivo1: descripción 1
archivo2: descripción 2
archivo2: descr. 2 (continuación)
archivo3: descripción 3

La primera línea indica una lista con todos los ficheros disponibles en el disco de la serie. El resto de líneas relacionan cada fichero de la lista anterior con una descripción del mismo, quizá en varias líneas.

Los ficheros "tagfile" tienen un contenido compuesto por comentarios y descriptores de archivos. Los primeros comienzan por el carácter "#". Los descriptores son líneas, una por cada fichero tgz de la serie, con el formato:

nombre: OPCION

Siendo "OPCION" una de cuatro posibilidades: ADD, REQ SKP y OPT. ADD indica que el fichero es imprescindible para la instalación, y se le incorpora a nuestro disco duro sin preguntarnos. Los ficheros REQ y OPT son parecidos entre sí. Para ambos, durante la instalación se nos consultará si deseamos incluirlos. La diferencia entre ellos estriba en el nivel de importancia para el correcto funcionamiento del sistema. Los primeros son "recomendados" (importantes) y los segundos son "opcionales" (poco importantes). Por último, los ficheros SKP no se instalan nunca, y son silenciosamente ignorados. Por supuesto, no hay ningún fichero marcado de esta manera en la instalación por defecto, pero nos puede resultar útil para una configuración a medida.

Cuando son seleccionados, setup instala los ficheros tgz siguiendo los siguientes comandos (supuesto montados en /dosa):

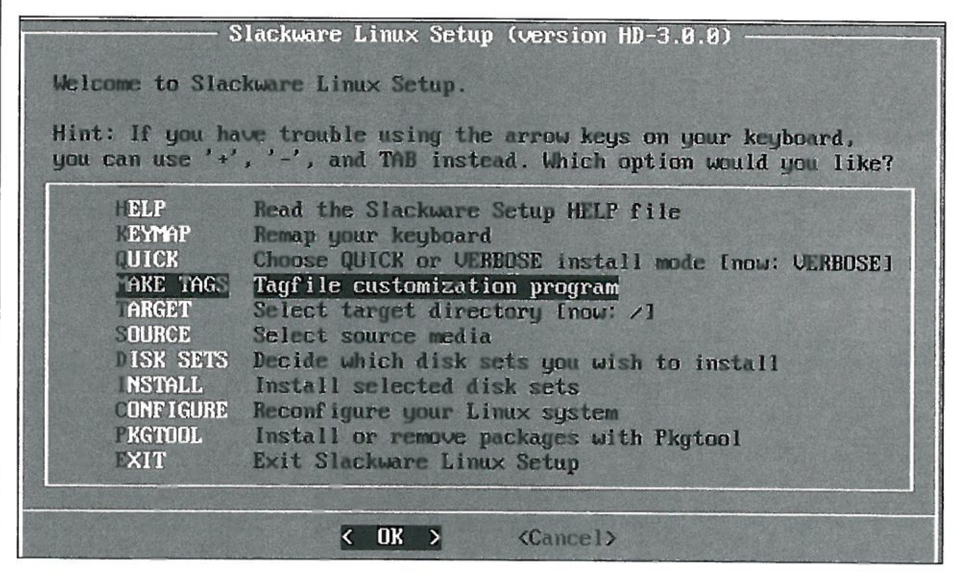


Figura 2: El menú principal de "setup".

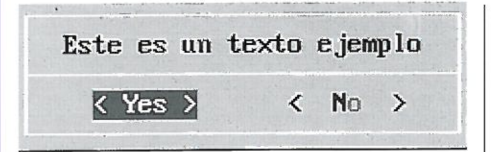


Figura 3: Diálogo tipo "yes/no"

cd /
tar xvfz /dosa/paquete.tgz

y, si se encontraba entre los ficheros descomprimidos, se ejecuta el fichero de configuración de cada paquete:
sh install/doinst.sh

UN EJEMPLO

Supongamos que deseamos crear una serie nueva, S, con cinco ficheros, f1.tgz (70 Kb), f2.tgz (395 Kb), f3.tgz (902 Kb), f4.tgz (130 Kb) y f5.tgz (254 Kb). Si sumamos el tamaño de los mismos, y pensamos en la limitación de 1.44 Mbytes por disco, llegamos a la conclusión de que deberemos repartir la serie en dos discos. Los ficheros asociados a la instalación serían:
Ejemplo del contenido de "disks1":

CONTENTS: f1 f2 f3
f1:
f1: Contenido de ejemplo del disco f1.
f1:
f2:
f2: Contenido ejemplo de f2
f2:
f3:
f3: Otro fichero, el f3
f3:

El contenido de "disks2" sería similar, con los ficheros f4 y f5:

CONTENTS: f4 f5
f4:
f4: Descripcion de f4
f4:
f5:
f5: El ultimo fichero
f5:

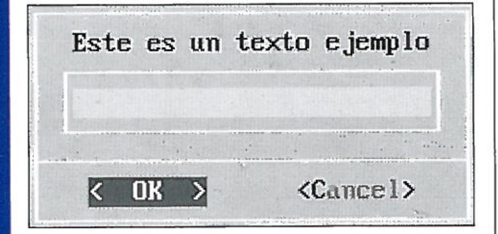


Figura 4: Diálogo tipo "inputbox"

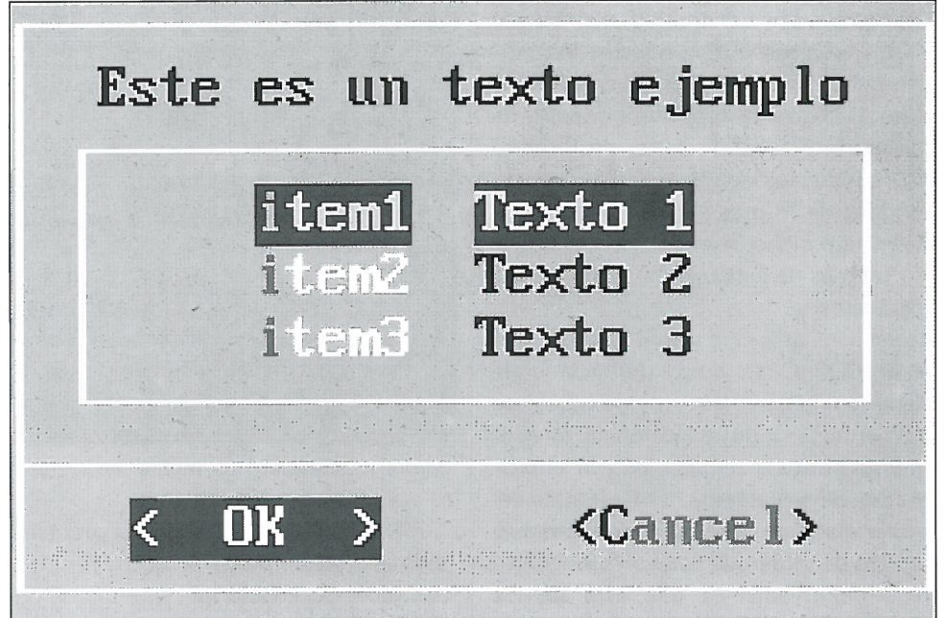


Figura 5: Diálogo tipo "menu".

Supongamos que consideramos obligatoria la instalación de f2 y f3, recomendada la de f1 y f4, y que f5 es opcional. El fichero "tagfile", en el primer disco, tendría el siguiente contenido:

Esto es un comentario
f1: REQ
f2: ADD
f3: ADD
f4: REQ
f5: OPT

Finalmente, supuesto que tuviesemos los contenidos de toda la serie en un servidor de disco, el listado de ficheros sería el siguiente:

s1/tagfile
s1/disks1
s1/f1.tgz
s1/f2.tgz
s1/f3.tgz
s2/disks2
s2/f4.tgz
s2/f5.tgz
s2/install.end

El fichero "maketag" sólo sería necesario si quisiéramos disponer de la opción para reconfigurar la serie S desde el programa de setup, y en nuestro caso podemos pasarlo por alto.

LA UTILIDAD "DIALOG"

Una vez vista la organización de los discos de las series, vamos a analizar unos pocos aspectos sobre el disco de instalación. En primer lugar, hemos comentado que está en formato minix. Este sistema de ficheros es el que se empleó en las primeras etapas de Linux, y todavía aparece en los discos de instalación debido a su simplicidad. Podemos acceder al contenido de este disco (supuesto insertado en la unidad A:) con:
mount -t minix /dev/fd0 /mnt

A partir de aquí podremos consultar o modificar lo que deseemos en el disco de instalación. Debemos tener en cuenta que el tamaño del mismo es muy limitado, para realizar posibles modificaciones en él. Cuando acabemos, podremos desmontar el disco de la forma tradicional,
umount /mnt

El programa más interesante de la instalación es el propio "setup". Consiste en un grupo de ficheros de comandos (ficheros batch, o de proceso por lotes), en el directorio /usr/lib/setup, que automatizan el proceso de instalación. Lo que da un aspecto más atractivo a dicha instalación es una utilidad, dialog, que se encarga de producir las pantallas que el programa setup utiliza. Sería tema de otro artículo la creación de un fichero batch, puesto que requiere el dominio de aspectos específicos de cada shell, y queda fuera del ámbito que estamos tratando hoy. Por contra, podemos examinar algo más a fondo la generación de pantallas, para aprovecharlas en nues-



Figura 6: Diálogo tipo "checkbox".

tras propias utilidades de instalación.

Este programa se encuentra disponible como /bin/dialog, si tenemos Linux instalado. Todas las formas de invocar la utilidad son:

- Diálogo para aceptar o rechazar un texto. Se muestra un texto en pantalla, y aparece un boton con la leyenda "yes" y otro con "no".
- Diálogo de mensaje, con pausa. Se muestra un texto en pantalla, y se proporciona un botón para aceptar.
- Diálogo informativo, sin pausa. Se muestra un texto en pantalla, y se retorna sin ninguna espera.
- Entrada de texto. Se proporciona una caja donde introducir texto, con scroll.
- Salida de texto. Se proporciona una caja para revisar el contenido de un fichero, con posibilidad de desplazamiento con cursores.
- Presentación de menú. Se muestra una lista donde cada elemento tiene dos componentes: identificativo y cuerpo. Se puede elegir un elemento cualquiera mediante cursores, o con una letra que representa a cada identificativo de forma única.
- Presentacion de lista de puntos. Se muestra una lista donde se puede activar o desactivar cada elemento por separado (checklist).

Cada opción admite la presentación de ventana con título opcional. Devuelven un estado de 0 cuando se selecciona un botón de "ok" o "yes", de 1 con "no" o "cancel", y -1 en cualquier otro caso. Un ejemplo de uso de cada aspecto, se muestra a continuación:

```
MSG="Este es un texto ejemplo"
TX=30
TY=5
dialog —yesno "$MSG" $TY $TX
dialog —msgbox "$MSG" $TY $TX
dialog —infobox "$MSG" $TY $TX
dialog —inputbox "$MSG" 8 $TX 2>out
dialog —textbox fichero $TY $TX
dialog —menu "$MSG" 10 $TX 3 \
        item1 "Texto 1" \
        item2 "Texto 2" \
        item3 "Texto 3"
dialog —checklist "$MSG" 11 $TX 4 \
        item1 "Opcion 1" on \
        item2 "Opcion 2" off \
        item3 "Opción 3" on \
        item4 "Opcion 4" off
```

En los ejemplos, TX y TY representan el tamaño horizontal y vertical de la ventana de representación, respectivamente. Los comandos de varias líneas se representan por una barra inclinada al final de cada línea incompleta, para facilitar la claridad de lectura. En las opciones de menú y lista, aparece un número que indica el número de posibilidades, y a continuación vienen los elementos de la lista, como se puede observar. El aspecto de la pantalla con estos comandos se pueden observar en las figuras que acompañan a este texto.

Finalmente, podemos comentar que se puede añadir título a la ventana de cualquiera de los ejemplos anteriores, como se muestra aquí:

```
dialog —title "Titulo" \
        —yesno $ETIQ $TY $TX
```

Y que es posible limpiar la pantalla tras concluir la misión de cada ventana, con:

```
dialog —clear —yesno $ETIQ $TY $TX
```

CONCLUSIÓN

En este artículo hemos aprendido la estructura básica de las distribuciones Slackware en Linux. Aunque puede resultar laborioso modificar una serie estándar para ajustarla a nuestros propósitos, podemos ahorrar trabajo en el futuro, en caso de que realicemos instalaciones frecuentes en varias máquinas. Por otro lado, incluso el usuario de casa puede solucionar posibles incorrecciones en una distribución, y realizar una instalación "manual", con un mínimo de conocimientos.

También, la existencia de dialog puede aportar vistosidad a nuestras propias presentaciones y utilidades, con muy poca programación. El mismo programa setup es una ingeniosa combinación de dicha utilidad.

En definitiva, esperamos haber desvelado otro apartado de este sorprendente sistema operativo. Hasta otro mes.

LA NUEVA VERSIÓN 5.3

Juan Manuel y Luis Martín

Dada la reciente aparición en el mercado de la nueva versión 5.3 del lenguaje Clipper, se ha considerado conveniente hacer un pequeño inciso en el curso que se estaba desarrollando con el fin de comentar los nuevos avances incorporados al lenguaje.

Hace ya bastantes meses se abrió un debate entre la comunidad de programadores en Clipper sobre la nueva versión del lenguaje. Computer Associates decidió someter a debate las principales mejoras que los programadores esperaban obtener en la nueva versión del lenguaje. Es conveniente resaltar que, aunque no se han incorporado todas las mejoras sugeridas, la cantidad y calidad de las mismas hacen que prácticamente no se echen en falta las no incorporadas.

Las mejoras y avances introducidos atañen a todos los campos que abarca el lenguaje, así como las utilidades que lo acompañan. En cualquier caso, si hay algo que el programador puede echar en falta es la no entrada de lleno del lenguaje dentro del mundo de la Programación Orientada a Objetos. A pesar de ello, cualquiera puede darse cuenta que se sigue avanzando en esta dirección.

Como se ha mencionado anteriormente, la nueva versión del lenguaje supone una importante mejora en todos los ámbitos del lenguaje. Entre estas mejoras cabe destacar la ampliación del número de clases predefinidas, las mejoras en las clases ya existentes, la incorporación de una librería gráfica y la inclusión de nuevos controladores RDD, que permiten una mejor utilización de las bases de datos.

En cuanto a las utilidades externas que acompañan al lenguaje cabe destacar la aparición de un Entorno Integrado de Desarrollo, así como un nuevo enlazador que genera aplicaciones para funcionar en modo protegido, rompiendo así la barrera de los 640 Kbytes de memoria.

UNA NUEVA FILOSOFÍA

Algo que se esperaba desde hace ya tiempo en Clipper era la incorporación de un Entorno Integrado de Desarrollo (IDE) que permitiese al programador tanto escribir como compilar, ejecutar y depurar una aplicación sin necesidad de utilizar diferentes herramientas para cada una de estas operaciones.

Computer Associates se ha echo eco de dicha necesidad, e incluso ha ido un poco más allá. Como se ha mencionado, la nueva versión 5.3 incorpora un IDE, pero con unas características un tanto especiales. Tal vez, la más llamativa sea que dicho entorno funciona bajo Windows. Esta característica no debe llevar a la confusión del programador. El lenguaje Clipper sigue siendo un lenguaje para creación de aplicaciones DOS y no Windows, aunque el IDE es una aplicación Windows. La elección de esta característica esta relacionada con el gran avance del propio entorno Windows, así como la facilidad de utilización que suponen las aplicaciones desarrolladas en entornos gráficos.

La utilización de dicho entorno no es ni mucho menos obligatoria y, aquellos programadores que deseen continuar trabajando en la forma clásica pueden seguir haciéndolo sin necesidad de realizar cambios en el programa. La variación viene para aquellos programadores que decidan utilizar el nuevo IDE.

El nuevo IDE trabaja de forma similar a como lo hacía la utilidad RMAKE,

es decir, un conjunto de ficheros entre los cuales se encuentran ficheros fuentes, librerías y ficheros de recursos. Esto supone una mayor modularidad en la forma de trabajo, facilitando tanto la concepción de las aplicaciones como el mantenimiento de las mismas.

LA LIBRERÍA GRÁFICA

Otro de los aspectos que se echaba de menos en un lenguaje como Clipper era la posibilidad de incorporar Gráficos en la aplicaciones. Sin embargo, en la nueva versión 5.3 se incluye una librería que permite la utilización del modo gráfico de la pantalla. Dicha librería tiene una serie de peculiaridades.

En primer lugar hay que destacar que la librería de gráficos no ha sido desarrollada por Computer Associates, sino por la empresa @@@. Debido a ello, la librería que se proporciona con el paquete no es la librería completa, aunque se incluyen gran cantidad de funciones que permiten una utilización del modo gráfico realmente potente.

Tal vez, la limitación más importante con la que se va a encontrar el programador es la imposibilidad de utilizar diferentes modos gráficos. Clipper 5.3 únicamente permite la utilización del modo VGA de 640x480 con 16 colores. Además, es imprescindible que la tarjeta gráfica sea 100% compatible con el estándar VESA 1.2 para tarjetas VGA. De cualquier forma, esta última limitación es posible subsanarla mediante la inclusión de una utilidad que permite simular dicho estándar por software. Dicha utilidad se encuentra en el directorio *\CLIP53\BIN* y se denomina *UNIVESA*.

La utilización de los gráficos dentro de una aplicación no supone grandes cambios en cuanto a la forma de trabajo tradicional. Indistintamente del modo de pantalla que se esté utilizando (gráficos o texto), es posible utilizar los mismos mandatos y funciones para visualizar la información en la pantalla. La única diferencia radica en las coordenadas que se especifican en los mandatos, ya que en el modo gráfico, dichas coordenadas van expresadas en pixels, y no en caracteres como en el modo texto.

Además, se han incorporado a la librería una serie de funciones especiales que permiten al programador la utilización de figuras geométricas dentro de la aplicación, con el fin de que la misma resulte aún más vistosa si cabe. También se incluyen funciones que posibilitan el manejo de ficheros gráficos, tanto para su lectura como su escritura, así como para la presentación de los mismos en la pantalla.

Otra de las cuestiones importantes dentro de los gráficos es el tratamiento del texto. La nueva librería incorpora funciones para el manejo de las diversas fuentes. De esta forma, se posibilita la carga de distintas fuentes, así como su manipulación, permitiendo visualizar el texto en negrita, cursiva o con distintos tamaños.

Por último, también se incluyen nuevas funciones para el tratamiento de la pantalla en modo gráfico que permiten crear zonas de exclusión en la pantalla, así como la generación de sombras y otros efectos realmente vistosos, todo ello con una facilidad realmente sorprendente.

LOS ENLAZADORES

CA-Clipper 5.3 se suministra con dos enlazadores que permiten generar el código ejecutable de las aplicaciones. El primero de ellos es una nueva versión de un viejo conocido de los programadores de Clipper; se trata del legendario *BLINKER*. Con este enlazador es posible generar aplicaciones que funcionan en el Modo Real. Sus principales características son las siguientes:

- Mayor velocidad de enlazado de las aplicaciones Clipper para el modo real.

- Posibilidad de utilización de *Overlays dinámicos* para el código de Clipper, C/C++ o Ensamblador de las aplicaciones desarrolladas con CA-Clipper 5.3, con el fin de sobrepasar la barrera de los 640 Kbytes de memoria.

- Mejora en la utilización del caché para la manipulación de los overlays.

- Compresión de la tablas de símbolos que posibilita una reducción de los requisitos de memoria y del tamaño del ejecutable.

Por otro lado, CA-Clipper 5.3 también se suministra con otro enlazador denominado *EXOSPACE*. Se trata de un enlazador capaz de crear aplicaciones que se ejecuten en el Modo Protegido. En dicho modo, la memoria extendida puede ser utilizada directamente. De esta forma, no es necesaria la utilización de overlays, ya que esta operación se realiza de forma automática, y solamente en aquellas ocasiones en que sea necesaria. La técnica empleada para el manejo del modo protegido es la misma que utilizan otras prestigiosas firmas como Lotus, IBM, Borland, etc. Sin el límite de los 640 Kbytes, todo el programa es cargado en la memoria, con lo que la velocidad de las aplicaciones mejora considerablemente.

Las mejoras y avances introducidos atañen a todos los campos que abarca el lenguaje

Lo mejor de este nuevo enlazador es que funciona de la misma forma que lo hacía el *RTLINK*, pudiendo incluso utilizarse los mismos ficheros Script.

CONTROLES VISUALES

Debido a la gran expansión alcanza por el entorno gráfico Windows, el modo de funcionamiento de las aplicaciones creadas para dicho entorno se ha convertido en todo un estándar dentro del mundo de la microinformática. Debido a ello, la mayoría de los lenguajes orientan sus herramientas de forma que el programador sea capaz de crear fácilmente aplicaciones con un aspecto lo más parecido posible a las este entorno. El propio Clipper también dirige sus pasos en este senti-

do, y lo hace mediante dos importantes innovaciones:

- La primera ha sido comentada anteriormente, y se trata de una librería de gráficos que permite al programador crear aplicaciones mucho más vistosas con un menor esfuerzo.

- La segunda consiste en una serie de clases predefinidas que dotan al lenguaje de una serie de controles muy utilizados en las aplicaciones Windows.

Estos controles suponen una importante ampliación en la Programación Orientada a Objetos de un lenguaje como Clipper. Además de aportar al programador una herramienta para la creación de aplicaciones mucho más vistosas, su uso resulta de una facilidad extraordinaria. De hecho, pueden ser utilizadas como una extensión de los mandatos *@...GET* y *MENU TO*. Mediante la utilización de estas clases, el programador será capaz de incluir controles tales como botones de comando, listas o menús desplegables, utilizando un solo mandato.

LAS NUEVAS CLASES

CA-Clipper 5.3 ha ampliado el número de clases predefinidas que pueden utilizarse en las aplicaciones. Se trata de nuevas clases que vienen a sumarse a las ya populares *Get* y *TBrowse*, y se encuentran claramente orientadas a la creación de interfaces más vistosos, elegantes y fáciles de utilizar para el usuario. Las nuevas clases incluidas en esta versión son las siguientes:

- La clase *CheckBox*: Permite crear objetos que representan "casillas de verificación". Estas casillas permiten al usuario seleccionar entre dos valores de tipo Verdadero o Falso (On y Off). Además, esta clase esta diseñada para ser utilizada de la misma forma que los objetos de la clase Get.

- La clase *ListBox*: Permite crear objetos que representan "Listas de elementos". De esta forma, permiten visualizar una serie de valores entre los cuales el usuario puede seleccionar uno de una forma fácil y cómoda.

- La clase *PushButton*: Permite crear objetos que representan "Botones de comando". Estos botones son capaces de llevar a cabo una operación cada vez que el usuario los pulse. Este tipo de objetos son frecuentemente utilizados en las aplicaciones Windows.

- La clase *RadioButton*: Permite crear objetos que representan "Botones de opción". Este tipo de objetos suele utilizarse en grupos, permitiendo seleccionar al usuario un solo un valor del grupo. Por tanto, los valores seleccionados resultan mutuamente excluyentes.

- La clase *RadioGroup*: Permite crear objetos mediante los cuales se pueden crear conjuntos de controles. Los objetos de esta clase se utilizan generalmente para agrupar objetos de la clase RadioButton.

Destacar la aparición de un entorno integrado de desarrollo y un nuevo enlazador

- La clase *ScrollBar*: Permite crear objetos que representan "Barras de Desplazamiento". La utilización de este tipo de objetos permite al usuario visualizar o acceder a los datos que se encuentran fuera del límite de una ventana, mediante el desplazamiento del contenido de la misma.

Además de estas clases, que proporcionan objetos que representan controles visuales, CA-Clipper 5.3 también incorpora otra serie de clases predefinidas para la creación y manipulación de menús. Esta nueva forma de trabajo con menús resulta mucho más elegante y eficiente que la tradicional, que como el lector podrá recordar, se realizaba mediante los mandatos @...PROMPT y MENU TO, así como la función ACHOICE(). Las nuevas clases encaminadas a la manipulación de menús son las siguientes:

- La clase *MenuItem*: Permite crear objetos que representan los elementos u opciones de un menú desplegable. Esta clase define los elementos básicos para el trabajo con las demás clases encaminadas al trabajo con menús. Esta forma de funcionamiento es similar a lo que ocurría hasta ahora con la clase TBColumn.

- La clase *PopUpMenu*: Permite crear objetos que representan menús desplegables o "PopUp Menus". Este tipo de menús están compuestos por uno o varios objetos de la clase anterior, que serán los elementos u opciones del menú desplegable. Además, los objetos de esta clase componen los elementos básicos para la creación de menús de barra.

- La clase *TopBarMenu*: Permite crear objetos que representan menús de barra. Los objetos de esta clase son los típicos menús que aparecen en la línea superior de la pantalla en la mayoría de aplicaciones. Estos menús están compuestos por menús desplegables, es decir, por objetos de la clase anterior.

LAS MEJORAS EN LAS CLASES GET Y TBROWSE

Como se ha mencionado anteriormente, en la nueva versión 5.3 del lenguaje no sólo se han incluido nuevas clases predefinidas, sino que además se han incorporado una serie de mejoras en las ya existentes.

El sistema *Get* se ha visto mejorado en varios aspectos, entre los cuales cabe destacar:

- Soporte para la utilización de ratón.

CORREO LECTORES

Para cualquier duda referente a los temas/artículos tratatos en la revista, escriba una carta a:

Sólo Programadores
Referencia: *Correo Lectores*
TOWER COMMUNICATIONS
C/ Marqués de Portugalete, 10 Bajo
Madrid 28027

- Soporte para la utilización de teclas rápidas.

- Inclusión de una barra de mensajes o estado.

- Facilidades para la actuación conjunta con los objetos de las clases anteriormente descritas.

- Ampliación del número de funciones del sistema Get, así como un mayor número de métodos en los propios objetos.

Por su parte, en la clase *TBrowse* se han incorporado varias propiedades nuevas que permiten definir el borde del objeto. También se proporciona un control mucho más exhaustivo del teclado, así como soporte para la utilización del ratón.

Por último, cabe destacar la inclusión de un modo de funcionamiento conjunto de ambas clases, permitiendo la edición de campos directamente desde el propio objeto *Browse*.

EL RATÓN

Por si las anteriores herramientas fueran pocas, junto con la librería de gráficos se han incluido una serie de funciones y constantes predefinidas que permiten utilizar dispositivos apuntadores como el ratón en las aplicaciones creadas con CA-Clipper 5.3.

En realidad, se trata de una serie de funciones incluidas en la librería gráfica que permiten al programador cargar un cursor determinado para el ratón, moverlo por la pantalla, detectar la pulsación de alguno de sus botones u obtener la posición en la que se encuentra y en la que se ha pulsado algún botón.

NUEVOS RDD's

El último avance incorporado a la versión 5.3, aunque no por ello sea el menos importante, es la ampliación del sistema de controladores sustituibles de bases de datos o RDD. La ampliación de este sistema se ha realizado en dos frentes. Por un lado se han realizado importantes mejoras en los controladores existentes. Por otro, se han añadido nuevos controladores.

Se incluyen nuevas funciones para el tratamiento de la pantalla en modo grágico

En lo referente a la mejora de los controladores ya existentes, cabe destacar la actualización de los algoritmos empleados por los distintos sistemas de manipulación de índices. En la versión 5.2 del lenguaje, los controladores ofrecían un funcionamiento mucho más lento en el sistema de índices del que realizan las aplicaciones nativas de dichos sistemas de índices.

Por otro lado, cabe destacar el aumento de compatibilidad entre los propios controladores y las aplicaciones nativas de los mismos. Esta mejora redunda en la eliminación de problemas existentes con el sistema de bloqueos cuando, trabajando en un entorno de red, se accedía de forma concurrente a una aplicación creada con Clipper que utilizaba estos controladores y la aplicación nativa del dicho controlador.

En lo concerniente a la inclusión de nuevos RDDs, éstos van especialmente encaminados a una utilización mucho más racional y versátil de los archivos de los campos Memo. Los nuevos controladores incluidos son los siguientes:

DBFMEMO. Este controlador permite la utilización y el mantenimiento de ficheros para campos Memo de forma compatible con otros controladores que, simultáneamente, pueden encargarse de la manipulación de los ficheros .DBF y de índice. De hecho, este controlador debe utilizarse conjuntamente con otro. De esta forma, el otro controlador se encarga de la gestión del fichero .DBF y de los ficheros de índice, mientras que el controlador DBFMEMO se encarga de la gestión de los ficheros de campos Memo. Las principales características que aporta este nuevo controlador en lo referente a los ficheros de campos Memo son:

- Limitación de tamaño a 4.2 GBytes.

- Bloques de tamaño mínimo de 1 Byte.

- Técnica especial de reciclaje del espacio libre del fichero.

- Capacidad de almacenamiento de cualquier tipo de datos de Clipper, incluidos los Bloques de Código.

DBFBLOB: Este controlador esta especialmente diseñado para proporcionar un mecanismo alternativo en el almacenamiento y mantenimiento de la información de los campos Memo. Entre sus principales características cabe destacar la posibilidad de utilizar ficheros de campos Memo sin necesidad del fichero .DBF al que tradicionalmente debían ir asociados. Esta característica lo hace idóneo para almacenar información de inicialización de las aplicaciones, tal como claves de acceso de usuarios, opciones configurables de las aplicaciones, configuraciones de color, etc.

CONCLUSIÓN

Con esta nueva versión de Clipper, Computer Associates incorpora numerosas mejoras en las herramientas existentes en el lenguaje. Además, se añaden otras nuevas herramientas que encaminan al lenguaje hacia el mundo de la Programación Orientada a Objetos y a los interfaces de usuario gráficos. Desde el punto de vista del usuario las aplicaciones desarrolladas con CA-Clipper 5.3 pueden llegar a ser mucho fáciles de manejar y mucho mas "amigables".

JUEGOS CONVERSACIONALES

Ignacio Cea

En todos los artículos que hemos presentado sobre C++, jamás se han planteado problemas complejos como el de guardar un objeto dentro de un fichero plano. Los ejemplos descritos sólo pretendían fijar las ideas que se describían en el texto adyacente y, por tanto, no podían ser excesivamente complicados.

Ahora, por el contrario, estamos jugando con un problema real y, en consecuencia, se plantean muchos problemas. Uno de ellos y, quizás el más típico bajo C++, es el de poder almacenar objetos en ficheros planos.

No hay que olvidar que dentro del mercado existen ya muchas casas que proporcionan librerías de objetos C++ que, con mayor o menor éxito solucionan este problema en bases de datos relaciónales. Sin embargo, éste es un recurso caro y no siempre a disposición de todos los programadores que han de sustituirlo por el versátil fichero plano. El siguiente artículo trata de buscar, bajo la carpa del desarrollo de un juego conversacional, una solución a ese problema.

Pero, antes de empezar es mejor que tomemos conciencia de cual es el mejor momento dentro del ciclo de vida de una aplicación para plantearnos este problema.

ANÁLISIS VERSUS DISEÑO

Las ideas planteadas a lo largo de las anteriores entregas de esta serie nos ayudan a obtener una visión general del problema que se trata de resolver.

Toda la exposición ha versado siempre sobre quienes son las entidades que participan en el problema y como se relacionan entre ellas para solucionarlo.

Sin embargo, no se ha mencionado prácticamente nada sobre la forma en la que dichos elementos han de ser implementados en lenguaje C++ sobre una máquina MSDOS. De hecho, tan sólo tendrá constancia de ello por los apuntes de código que acompañaban a cada artículo y cuya única intención era orientar un poco más sus pasos. Si no fuera por este motivo, no habría necesidad de haberlos puesto.

Observado el diagrama no encontrará respuestas a preguntas tan típicas como: ¿Cuántas instancias de la clase "Objeto del Juego" van a existir?, ¿Quién las crea?, ¿Dónde se almacenan?, etc...

El análisis se centra siempre en la primera parte.

DISEÑO VERSUS ANÁLISIS

En la etapa de diseño el modelo de objetos inicial se detalla más para permitir su implementación en una máquina concreta, con unos recursos determinados y bajo un lenguaje de programación dado.

El hecho de tomar decisiones relativas a la implementación del modelo suele traducirse en la primeras etapas, a su vez, en la aparición de nuevas clases que ya no son del problema sino de la solución. Esas nuevas clases han de seguir también todo el ciclo de análisis, diseño e implementación.

Este artículo hablará sobre el diseño de la persistencia de objetos.

VELOCIDAD Y ACCESO DIRECTO

A lo largo de las primeras entregas del curso de C++ se estudió una forma en la que el contenido de los objetos podía grabarse dentro de un fichero. El método pasaba por la sobrecarga del operador de inserción, tal y como muestra en la figura 1.

Sin embargo, los ficheros así generados son secuenciales y, por tanto, para poder recuperar los datos de un objeto cualquiera ha de recorrer antes todos los que le preceden. No parece este el

Es evidente que a lo largo de los capítulos anteriores no se ha hablado nada sobre la manera en la que han de solucionarse los muchos problemas que acarrea llevar a una máquina el modelo de objetos. Siga este artículo y aprenderá a resolver el de la persistencia de objetos bajo ficheros.

FIGURA 1

```
class ClasePrueba
{
  private :
    int A;
  public :
    ClasePrueba () : A (0) { /* Vacío */ }
    ClasePrueba (int n): A (n) { /* Vacio */ }
    friend ostream &operator << (ostream &out,
const ClasePrueba &CA)
                        { return (out << CA.A); }
          friend  istream  &operator  >>  (istream
&int,ClasePrueba &CA)
                        { return (in >> CA.A); }
};

void main (void)
{
  ofstream FileOut ("Fichero.txt");
  for (int i = 0;i < 10;i++) {
    ClasePrueba C (i);
    FileOut << C << endl;
  }

  ifstream FileIn ("Fichero.txt");
  for (int j = 0;j < 10;j++) {
    ClasePrueba C ();
    FileIn >> C;
  }
}
```

Sobrecarga del operador de inserción.

FIGURA 2

```
#include "fstream.hh"
extern "C" {
  #include <string.h>
};

class Persona
{
  private :
    char Nombre [50
    unsigned Edad;
    float Altura;
  public :
    Persona (char *n,unsigned e,float a)
                  : Nombre (new char [strlen (n) +
1),Edad (e),Altura (a) { strcpy (Nombre,n); }
    ~Persona () { delete Nombre }
    friend ostream &operator (ostream &out,
const Persona &p)
{ return (out << p.Nombre << " " << p.Edad
<< " " << p.Altura); }
};

void main (void)
{
          Persona Autor ("Ignacio Cea",26,1.80);
          cout << Autor << ": " << sizeof (Autor)
<< "bytes" << endl;
}
```

Sizeof de una clase sencilla.

camino adecuado para conseguir velocidad de ejecución.

Para conseguirlo hemos de hacer que todos los objetos tengan el mismo tamaño y grabarlos dentro de ficheros binarios. Bastaría, en ese caso, con saber la posición que dentro del fichero ocupa el objeto requerido y, con sólo multiplicar por el tamaño del mismo, acceder al byte donde comienza.

FIGURA 3

```
#include "fstream.hh"
extern "C"
{
  #include <string.h>
}

class String
{
  private :
    char *Texto;
    unsigned Lon;
  public :
    String (char *t) : Texto (new char [strlen (t) + 1),
Lon (strlen (n) { strcpy (Texto,t); }
    String (const String &s) : Texto (new char
[s.Lon + 1), Lon (s.Lon) { strcpy (Texto,s.Texto); }
    ~String () { delete Texto; }
    friend ostream &operator << (ostream &out,
const String &s)
{ return (out << Texto); }
};

class Persona
{
  private :
    String Nombre;
    unsigned Edad;
    float Altura;
  public :
    Persona (const String &n,unsigned e,float a)
            : Nombre (n),Edad (e),Altura (a)
{ /* Vacio */ }
    ~Persona () { /* Vacio */ }
    friend ostream &operator (ostream &out,
const Persona &p)
                        { return (out << p.Nombre <<
" " << p.Edad << " " << p.Altura); }
};

void main (void)
{
          String nombre ("Ignacio Cea");
          Persona Autor (nombre,26,1.80);
          cout << Autor << ": " << sizeof (Autor) <<
"bytes" << endl;
}

// Este programa necesita una clase String que
soporte...
// ...el manejo de cadenas. Esta clase se sumistrará
en posteriores capítulos
```

Una clase compleja.

Esta es la razón de ser de operaciones como "*fwrite*" y "*fread*" que todos hemos usado para grabar estructuras de datos (struct) en el vetusto lenguaje C. Las preguntas que surgen tras este apunte son: ¿Puede hacerse lo mismo con objetos sabiendo que estos además de datos llevan métodos?. ¿Guarda el compilador como parte de los atributos de una clase información sobre sus métodos?. ¿Qué pasa si esa información la cargo de un disco?.

GRABANDO UNA CLASE SENCILLA

Pensemos en un principio en una clase sencilla. Por sencilla se entiende que tiene sólo atributos pertenecientes a tipos básicos (char [],int, long, etc...) y que no hay declarado ningún método virtual. Consideramos también, inicialmente, que no hay atributos de tipo puntero.

FIGURA 4

```
class ClasePrueba
{
  private :
    int A;
  public :
    ClasePrueba (int n) : A (n) { /* Vacio */ }
    virtual void dibujar (void) { cout << A << endl; }
};

void main (void)
{
  ClasePrueba C (3);
  C.dibujar ();
  cout << "Tamaño: " << sizeof (ClasePrueba)
<< " bytes" << endl;
}
```

Los métodos virtuales.

FIGURA 5

```
class ClasePuente

{

  private :

    // ...Atributos de la clase original pasados a tipos
básicos


  public :

    // Métodos de conversión directa e inversa

    ClasePuente (const ClaseOrigen &);

    ClaseOrigen *generar (void);

}
```

Una clase puente.

En tal caso podríamos afirmar que, por un lado, el sistema no guarda dentro de los atributos de las clase información alguna sobre ninguno de sus métodos y que, por otro, todas las instancias creadas a partir de ella tienen siempre el mismo tamaño. Le invitamos a que lo compruebe mediante el operador "*sizeof*" de C++ y el ejemplo que se muestra en la figura 2.

En tal caso, podemos considerar a la clase como equivalente, desde el punto de vista de persistencia, a las estructuras de C y emplear métodos equivalentes al "*fwrite*" y "*fread*" para llevarla a cabo.

¿Y SI HAY ATRIBUTOS NO BÁSICOS?

Imagine ahora que nuestra clase piloto contiene atributos pertenecientes a otras clases (Figura 3). Si esas otras clases siguen las normas definidas en

FIGURA 6

```
//=======================================
================================//
// CLASE : FicheroEstructura
//
// DESCRIPCION: Clase para el almacenamiento de estructuras en disco //
// AUTOR : Ignacio Cea Forni,s
//
//=======================================
================================//

#if !defined (__STRFILE_HH__)
#define __STRFILE_HH__

#include "String.hh" // Por el nombre de fichero
#include <fstream.h> // Por empleo de canales

//————Defini¢n de la estructura de la clase————//
template<class T> class FicheroEstructura {
  public :
    // Constructores
    FicheroEstructura (const String &);
    virtual ~FicheroEstructura ();

    // Manipuladores de los objetos de la clase...
    void limpiar (void);
    long posicion (void) const;
    long size (void) const;
    const String &nombre (void) const;
    const T &objeto_apuntado (void) const;
    int esta_abierto (void) const;
    const T &primero (void);
    const T &ultimo (void);
    const T &siguiente (void);
    const T &anterior (void);
    int esta_vacio (void) const;
    int hay_mas (void) const;
    int sobreescribir (const T &);
    int insertar (const T &);
    void abrir (void);
    void cerrar (void);

    // Operadores
    operator void * (void);
    int operator ! (void);
    const T &operator [] (unsigned);
    const T &operator ++ (int);
    const T &operator — (int);

  protected :
    unsigned char *Buffer;// Lugar donde se almacena el £ltimo objeto
    long Posicion;        // Posici¢n dentro del fichero en objetos para I/O
    long Size;            // Tama¤o del fichero en objetos...
    fstream Fichero;      // Canal de comunicaciones
    String Nombre;        // Nombre del canal de comunicaciones
    int Abierto;          // ¨ Est abierto o cerrado ?
};
//—————————————————————//

    // M,todos inline
template<class T> inline
FicheroEstructura<T>::~FicheroEstructura ()
{ delete [] Buffer; }
template<class T> inline long
FicheroEstructura<T>::posicion (void) const
{ return (Posicion); }
template<class T> inline long
 FicheroEstructura<T>::size (void) const { return (Size); }
template<class T> inline const String
&FicheroEstructura<T>::nombre (void) const
{ return (Nombre); }
template<class T> inline const T
&FicheroEstructura<T>::objeto_apuntado (void) const
        { return (*((T *) Buffer)); }
template<class T> inline int
FicheroEstructura<T>::esta_abierto (void) const
{ return (Abierto); }
template<class T> inline int
FicheroEstructura<T>::esta_vacio (void) const
{ return (!Size); }
template<class T> inline int
FicheroEstructura<T>::hay_mas (void) const
{ return ((Size == 0 || (Posicion + 1) == Size) ? 0 : 1); }
template<class T> inline void
FicheroEstructura<T>::cerrar (void)
{ if (Abierto) { Fichero.close (); Abierto = 0; } }
template<class T> inline
FicheroEstructura<T>::operator void * (void)
{ return (Abierto ? this : NULL); }
template<class T> inline int
FicheroEstructura<T>::operator ! (void) { return (!Abierto); }
template<class T> inline const T
&FicheroEstructura<T>::operator ++ (int)
{ return (siguiente ()); }
template<class T> inline const T
&FicheroEstructura<T>::operator — (int)
{ return (anterior ()); }

#endif

// Fin del fichero StrFile.hh
// Versi¢n 1.0
// 30 de Marzo de 1995
```

La clase FicheroEstructura.

el anterior punto, el problema no sería mayor que el de manipular una estructura que tiene definida otra dentro.

Sin embargo, si esto no es así los objetos de la clase no tienen porque tener un mismo tamaño y, por tanto no pueden ser grabados, con garantías de recuperación, dentro de un fichero binario.

Para solucionar el problema hemos de construir una clase adicional que haga de puente entre nuestro objeto y el fichero binario y que, por el contrario, sí reúna las características ideales apuntadas en el párrafo anterior. Seguro que ya se ha preguntado: ¿Por qué tener dos clases para hacer lo mismo?, ¿Por qué no prescindir de la compleja?. Su pregunta se resolverá más adelante.

MÉTODOS VIRTUALES

Volvamos a la clase ideal del punto anterior y añadámosle un método virtual. Esto, inmediatamente, hace pensar en una clase inmersa dentro de alguna estructura jerárquica.

El sistema guarda por cada método virtual un puntero al método que realmente será invocado cuando se le nombre. Por tanto si salvamos la clase, estaremos salvando, también el valor de esos punteros.

No olvidemos, sin embargo, que esas direcciones son asignadas al cargar el programa y que, por tanto, dependen del sistema operativo. En dos ejecuciones consecutivas del mismo programa pueden asignarse distintas direcciones de memoria.

Al cargar de disco un objeto se cargan también los valores de esos punteros que sustituirán a los que ya estaban. No hace falta que le digamos que tras esa operación invocar al método virtual puede originar un serio desastre. Compruebe ese tamaño aparente a través del ejemplo de la figura 4.

En este caso se hace imprescindible el empleo de clases puente.

TODOS LOS HIJOS AL MISMO SACO

Imagine, por otro lado, una jerarquía de clases ideales cuyas instancias van a ser todas grabadas dentro del mismo fichero. Sin el empleo de una clase puente esto no sería posible ya que ninguna de las clases hijas tiene, ni siquiera, el mismo tamaño y tanto más si, encima hay definidos métodos virtuales.

LAS CLASES PUENTE

Parece, en definitiva, evidente que la solución al problema de la persistencia de objetos pasa, en la mayor parte de las ocasiones, por la construcción de una clase puente que reúna las características de lo que hemos venido en denominar clase ideal; esto es, que no disponga de métodos virtuales y todos sus atributos sean de tipos básicos.

Esta clase debe disponer, además, de dos métodos: uno que sirva para crear objetos puente a partir de distintas fuentes y el otro para la operación inversa. (Figura 5)

EL JUEGO CONVERSACIONAL

Parece claro que dentro del modelo de objetos del juego conversacional presentado en el artículo anterior las únicas clases que van a tener persistencia son: "Objeto del Juego", "Personaje del Juego", "Lugar de Juego" y "Elemento de Comunicación" ya que de ellos pueden existir tantas instancias que no quepan en la memoria.

RECORRIENDO FICHEROS PUENTE

Cada una de las distintas jerarquías a las que esas clases dan lugar va a ser almacenada dentro de un fichero binario distinto y a través de una clase puente distinta.

El manejo de los ficheros va a ser, en todos los casos, con la salvedad del tipo de información que dentro de ellos se almacena. Por tanto, parece lógico el desarrollo de una clase *template* que reúna los métodos necesarios para navegar por dentro de un fichero binario de objetos puente. Dicha clase se ofrece en las figuras 6 y 7.

UNA LISTA EN DISCO

La clase se comporta como si de una lista de objetos se tratara con la única salvedad de que esa lista en lugar de estar en la memoria está en el disco. Por tanto, podemos hacer cosas como situarnos en un elemento concreto o navegar hacia adelante y hacia atrás por los elementos de la colección.

Como elementos de navegación fundamentales tiene los métodos: "primero", "ultimo", "siguiente" y "anterior" así como la sobrecarga del operador [] que permite posicionarnos directamente en un objeto concreto del fichero sin necesidad de pasar por todos los que le preceden dando así vida al concepto de acceso directo que preconizábamos en puntos anteriores.

Una forma típica de recorrer el fichero dentro de un bucle for podría ser la siguiente:

```
FicheroEsctructura <MiClase> Fichero ("Fichero.txt");
for (int i = 0;i < Fichero.size ();i++)
cout << Fichero [i] << endl;
```

suponiendo que, por un lado el fichero abierto ya contenga objetos del tipo MiClase y, segundo, que esté sobrecargado el operador de inserción dentro de aquella para permitir su visualización por pantalla.

MÉTODOS ADICIONALES

Como métodos adicionales están los que permiten añadir un nuevo elemento a la parte final del fichero (insertar) o el de sobreescribir el que esté actualmente señalado dentro de aquel con el que se recibe como parámetro (sobreescribir).

FIGURA 7

```
//=========================================================================//
// CLASE: FicheroEstructura                                   //
// AUTOR: Ignacio Cea Forniés                                              //
//=========================================================================//

#include "StrFile.hh"  // Prototipos de métodos y variables
extern "C" {
  #include <stdlib.h>   // Por exit...
  #include <stdio.h>    // Por unlink...
};

//————————Definición de la clase————————————————————————//
//=========================================================================//
// METODO     : Constructor (Ignacio Cea Forniés)        FECHA: 31/3/95 //
//=========================================================================//
//·DESCRIPCION: Crea una instancia de la clase FicheroEstructura //
//·PARAMETROS :    (const String &) Nombre del fichero a abrir           //
//·RETORNO    : Ninguno                                   //
//·CAMBIOS    : —                                    //
//·COMENTARIOS: —-                                                                   //
//=========================================================================//
template<class T> FicheroEstructura<T>::FicheroEstructura (const String &n)
: Buffer ((T *) NULL),
  Posicion (0),
  Size (0),
  Nombre (n),
  Abierto (0)                         // ...en principio
{

// Abre hueco para el buffer...
Buffer = new unsigned char [sizeof (T)];
if (!Buffer) exit (1); // ...no hay memoria para el buffer
for (int i = sizeof (T) - 1;i >= 0;*(Buffer + i) = 0,i—);

// Abre el fichero
abrir ();

}

//=========================================================================//
// METODO     : limpiar (Ignacio Cea Forniés)                  FECHA: 20/8/95 //
//=========================================================================//
//·DESCRIPCION: Elimina del fichero toda la información que contiene      //
//·PARAMETROS :     Ninguno                                  //
//·RETORNO    : Ninguno                                   //
//·CAMBIOS    : —-                                   //
//·COMENTARIOS: —-                                                                   //
//=========================================================================//
template<class T> void FicheroEstructura<T>::limpiar (void)
{

// Cierra el fichero y lo borra...
Fichero.close ();
unlink (Nombre);

// Establece las variables internas a su valor inicial
Posicion = 0;
Size = 0;

// Si estaba abierto, lo vuelve a abrir...
if (Abierto)
  Fichero.open (Nombre,ios::in | ios::out | ios::binary);

}

//=========================================================================//
// METODO     : primero (Ignacio Cea Forniés)                  FECHA: 31/3/95 //
//=========================================================================//
//·DESCRIPCION: Desplaza el puntero del fichero al primer elemento del  //
//              mismo, lo extrae y se le queda apuntano.               //
//·PARAMETROS :     Ninguno                                  //
//·RETORNO    : (const T &) Referencia al objeto capturado              //
//·CAMBIOS    : —-                                   //
//·COMENTARIOS: —                                       //
//=========================================================================//
template<class T> const T &FicheroEstructura<T>::primero (void)
{

// Si el fichero está cerrado, o no tiene nada...
// ...no puedo acceder a ningún elemento y retorno el que hay
if (!Abierto || Size == 0)
  return (*((T *)Buffer));

// Coloco los punteros de lectura y escritura al principio...
// ...carga los datos y los retorna al usuario
Posicion = 0;
Fichero.seekg (0l,ios::beg);
Fichero.seekp (0l,ios::beg);
Fichero.read (Buffer,sizeof (T));
Fichero.seekg (0l,ios::beg);          // Puntero sobre el primero...
return (*((T *) Buffer));

}

class ClasePuente
{
```




```
    private :
    //...Atributos de la clase original pasados a tipos básicos

  public :
    // Métodos de conversión directa e inversa
    ClasePuente (const ClaseOrigen &);
    ClaseOrigen *generar (void);
}

//=====================================================================//
// METODO     : ultimo (Ignacio Cea Forniés)            FECHA: 31/3/95 //
//=====================================================================//
//·DESCRIPCION: Desplaza el puntero del fichero al último elemento del  //
//               mismo, lo extrae y se queda marcando el final.        //
//·PARAMETROS :  Ninguno                                   //
//·RETORNO    : (const T &) Referencia al objeto capturado          //
//·CAMBIOS    : —-                                  //
//·COMENTARIOS: —                                                          //
//=====================================================================//
template<class T> const T &FicheroEstructura<T>::ultimo (void)
{

// Si el fichero está cerrado o no hay datos...
// ...retorno lo que haya dentro del buffer
if (!Abierto || Size == 0)
  return (*((T *)Buffer));

// Coloco los punteros de lectura y escritura al principio...
// ...carga los datos y los retorna al usuario
Posicion = Size - 1;
Fichero.seekg (-((long) sizeof (T)),ios::end);
Fichero.seekp (-((long) sizeof (T)),ios::end);
Fichero.read (Buffer,sizeof (T));
Fichero.seekg (-((long) sizeof (T)),ios::end);
return (*((T *) Buffer));

}
//...El resto está en el disco
```

Un trozo de la clase FicheroEstructura.

Un método más hace su aparición y es el de "limpiar" que permite eliminar del disco toda la información que el fichero contenga y dejarlo vacío.

La clase siempre mantiene siempre en memoria el objeto que ha sido manejado en última instancia y puede ser accedido a través del método objeto_apuntado. Hay que tener cuidado con el empleo de este método ya que cualquier operación sobre el fichero puede alterar el valor devuelto por este método e influir en el resultado de acciones posteriores.

AMPLIANDO LA CLASE

Probablemente usted, en el futuro, requerirá de esta clase muchos más servicios, por ejemplo, si está interesado en realizar búsquedas de objetos dentro del fichero u ordenaciones indexadas del mismo.

Eso puede usted conseguirlo fácilmente haciendo uso de la herencia. No tiene más que derivar su clase con funcionalidad añadida de ésta para seguir teniendo a su disposición toda la potencia que ahora se le da. A continuación se le muestra un ejemplo de como debería ampliarla para conseguir una clase que manejara sus propias estructuras y que incluyera un método para encontrar un objeto determinado dentro del fichero pasándole como parámetro el identificador del mismo.

```
class FicheroMio : public
FicheroEstructura <MiClase>
{
        ...
        public :
                ...
                const MiClase &buscar
(const String &);
                ...
};
```

BORRAR OBJETOS

Habrá caído ya en la cuenta de que la clase no posee ningún método que nos permita borrar alguno de los objetos grabados dentro del fichero. Dado que la clase no sabe que tipo de estructuras va a manejar puesto que se trata de un template el único método de borrar que estaría a nuestra disposición, de haberse implementado, sería el que reconstruye el fichero entero cada vez que se borra una clase. Esto no parece una solución viable sobretodo si pensamos en que el fichero puede llegar a contener gran cantidad de ficheros.

Se deja, por tanto, como responsabilidad del usuario y su herencia el definir la forma en la que se borran los objetos de un fichero.

UNA SOLUCIÓN TÍPICA

Dado que la clase está pensada, como se apunto al principio, para manejar objetos de clases tipo puente podemos siempre incorporar un atributo básico que nos diga si está o no borrada, junto con métodos para cambiar su estado.

```
class ClasePuente
{
        private:
                ...
                int borrado;
                ...
        public :
                ...
                int estaBorrado (void)
{ return (borrado); }
                void borrar (void);
                void undo (void);
                ..
};
```

Nuestra clase hija de la que se muestra como ejemplo debería, por un lado añadir un método para borrar un objeto que lo único que hiciera fuera buscarlo en disco, invocar al método borrar del objeto alcanzado y sobreescribirlo en el disco y, por otro, redefinir el destructor de la clase *FicheroEstructura* para que al finalizar reconstruyese el fichero ignorando todos aquellos objetos que tuvieran a uno el indicador de borrado.

ALGO MÁS

Habrá observado también que la clase FicheroEstructura hace uso dentro de otra llamada "String". De momento no sirve más que para ocultar cadenas char *, por lo que, incluso, puede hacer uso de la que aparece en la figura 3 o de la que se presento como parte del curso de C++ pasado.

LA PRÓXIMA ENTREGA

No pierda de vista esa clase ya que para las próximas entregas será algo fundamental ya que nuestro objetivo será llevar a disco las Clases "Objeto del Juego", "Lugar del Juego", "Elemento de Comunicación" y "Personaje".

INCLUSIÓN DE SONIDOS EN ARCHIVOS DE AYUDA

Enrique Castañón

Son pocas las carencias del sistema de ayuda de Windows, y siempre queda la opción "hágaselo usted mismo" todo lo que no tiene la ayuda de Windows lo podemos implementar mediante una librería de enlace dinámico.

Se van a explicar dos métodos para incluir sonidos en archivos de ayuda. El primero es el más simple, y no requiere ningún conocimiento de programación. El segundo es el de crear una Librería de Enlace Dinámico con archivos de sonido WAV como recursos, y con el código necesario para ejecutar estos archivos.

Con cualquiera de los dos métodos los resultados son similares, pero si optamos por crear una DLL, ésta nos será útil no sólo para crear hipertextos, sino que se podrá utilizar en cualquier aplicación que creemos para Windows. Se podrá utilizar la DLL como librería de sonidos de carácter general.

Este artículo será de utilidad tanto a los que pretendan saber cómo incluir sonidos en los archivos de ayuda, como a los interesados en la programación Windows. Se explicará además como crear Librerías de Enlace Dinámico con archivos de sonido WAV como recursos.

UNA MANERA SENCILLA DE INCLUIR SONIDOS

El primer paso es crear una macro de usuario, para ello basta con registrar la función del API de Windows *sndPlaySound*, de la que se darán detalles más adelante. Como se explicó en una pasada entrega de esta serie de artículos, para registrar una macro de usuario hay que incluir la macro *RegisterRoutine* en la sección CONFIG del archivo de proyecto, tal y como muestra la figura 1. El primer parámetro de la macro RegisterRoutine es el nombre de la DLL que contiene la función, en este caso la DLL se llama *MMSYSTEM*, no es necesario incluir la extensión, aunque no es un error hacerlo, esta librería se encuentra el directorio System de Windows. El segundo parámetro es el nombre de la función, *sndPlaySound*. Y el tercer parámetro es el formato de los parámetros de la función registrada, por ahora baste con decir que para el caso de *sndPlaySound* habrá que incluir "Su", las equivalencias de formato de parámetros con C se muestran el la figura 2.

Una vez registrada la función, se puede utilizar *sndPlaySound* como una macro más del sistema de ayuda. Para ejecutar un archivo de sonido WAV con *sndPlaySound*, como primer parámetro hay que indicar el nombre del archivo WAV, y en segundo lugar la forma en que se ejecutará el archivo, indicando 0, no se podrá seguir trabajando hasta que el archivo se termine de ejecutar, si indicamos 1, se podrá trabajar mientras el archivo se ejecuta, la figura 3 muestra la lista de todos los valores, y más adelante se explica su significado. El siguiente ejemplo ejecutaría el archivo musica.wav:

sndPlaySound("musica.wav",1). La figura 4 muestra el archivo RTF y el modo de usar *sndPlaySound* como macro.

LIBRERÍAS DE ENLACE DINÁMICO

Una Librería de Enlace Dinámico es un módulo ejecutable que puede contener funciones o recursos para ser utilizados por aplicaciones e incluso por otras DLLs. Las Librerías de Enlace

Dinámico son uno de los elementos más importantes de Windows. Los archivos KERNEL.EXE, GDI.EXE y USER.EXE son librerías de enlace dinámico, incluso los controladores DRV y los archivos de fuentes FON también lo son.

A diferencia de los ejecutables EXE las DLLs no tienen la función *WinMain*, en su lugar hay que suministrar la función *LibMain*. Windows llama una sola vez a *LibMain* cuando la DLL se carga en memoria por primera vez, el prototipo de *LibMain* es el siguiente:

int FAR PASCAL LibMain(HINSTANCE hInstance, WORD wDataSeg, WORD cbHeapSize, LPSTR lpCmdLine)

El primer parámetro, *hInstance*, es el handle de la DLL. *wDataSeg* es el valor del segmento de datos. *cbHeapSize* es el tamaño del heap local. *lpCmdLine* es un puntero a la línea de comando, este valor suele ser NULL debido a que normalmente las DLLs se cargan sin parámetros.

Las DLLs tiene que tener además de *LibMain* la función WEP, que es el punto de salida de la DLL. La función WEP es llamada por Windows cuando la DLL se descarga de memoria, el prototipo de WEP es el siguiente:

int FAR PASCAL WEP(int bSystemExit)

Los valores posibles de *bSystemExit* son: *WEP_SYSTEMEXIT* que indica que Windows se esta cerrando y *WEP_FREE* que indica que únicamente es la DLL la que se está descargando de memoria.

Todas las funciones que deseemos utilizar como macros en un archivo de ayuda deberán ser declaradas como *FAR PASCAL EXPORT*, si no se incluye EXPORT en la declaración de la función, deberá incluirse la función en la sección EXPORTS del archivo de definición de módulo. La figura 5 se muestra el código de la librería.

La función sndPlaySound del API de Windows, nos permite reproducir archivos de sonido WAV

El tema de las Librerías de Enlace Dinámico es amplio, y se podría escribir mucho sobre ellas. Dado que el tema del artículo es el hipertexto y no la programación en Windows, nos hemos limitado a dar una somera explicación que pueda ayudar a comprender el ejemplo que acompaña a este artículo.

COMO CREAR EL ARCHIVO DE RECURSOS

Los recursos son datos que normalmente no se modifican en tiempo de ejecución. Los cuadros de diálogo, los iconos, cursores, fuentes, etc., son recursos, que pueden residir en un ejecutable EXE o en una DLL. Además existe un tipo especial de recursos, los recursos definidos por el usuario, éste es el tipo de recurso que utilizaremos para añadir archivos WAV como recursos a la DLL, su sintaxis es la siguiente:

ID_RECURSO TIPO "ARCHIVO"

ID_RECURSO es el identificador o nombre del recurso, *TIPO* es el nombre del tipo de recurso y *"ARCHIVO"* es el nombre del archivo que contiene el recurso. Así es como quedará nuestro archivo fuente de recursos:

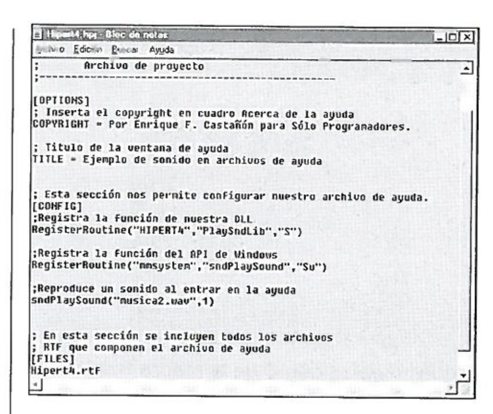
MUSICA1 SONIDO "MUSICA1.WAV"
MUSICA2 SONIDO "MUSICA2.WAV"

La extensión de un archivo fuente de recursos es RC, y deberá estar en texto ASCII. Una vez creado se puede compilar con la utilidad de Microsoft RC.EXE con la opción -r, para crear un archivo tipo RES, y posteriormente enlazarlo con la DLL, o directamente enlazar el archivo RC con la DLL de la siguiente manera:

RC ARCHIVO.RC ARCHIVO.DLL

COMO ACCEDER A LOS RECURSOS

Podemos crear una DLL de funciones o de recursos, o con funciones y recursos. Para el ejemplo que acompaña a este artículo, hubiese bastado con incluir solamente los archivos de sonido como recursos, sin embargo, incluyendo el código para manejar los recursos, conseguimos encapsular los datos junto al código que los trata, de esta manera, para la posterior utilización de la DLL sólo será necesario recordar la sintaxis de una sola función, que en nuestro ejemplo es *PlaySndLib(nombreSonido)*, sin tener que preocuparnos de implementar código para manejar el recurso.



```
;        Archivo de proyecto
;--------------------------------------------

[OPTIONS]
; Inserta el copyright en cuadro Acerca de la ayuda
COPYRIGHT = Por Enrique F. Castañón para Sólo Programadores.

; Titulo de la ventana de ayuda
TITLE = Ejemplo de sonido en archivos de ayuda

; Esta sección nos permite configurar nuestro archivo de ayuda.
[CONFIG]
;Registra la función de nuestra DLL
RegisterRoutine("HIPERT4","PlaySndLib","S")

;Registra la función del API de Windows
RegisterRoutine("mmsystem","sndPlaySound","Su")

;Reproduce un sonido al entrar en la ayuda
sndPlaySound("musica2.wav",1)

; En esta sección se incluyen todos los archivos
; RTF que componen el archivo de ayuda
[FILES]
Hipert4.rtf
```

Figura 1. El archivo de proyecto.

El API de Windows posee funciones especificas para cargar cada tipo de recurso, *LoadIcon*, carga un recurso de icono, *LoadCursor*, carga un recurso de cursor. Para cargar recursos definidos por el usuario se utiliza la función *LoadResource* conjuntamente con *FindResource*. En el ejemplo que acompaña este artículo, y por razones de claridad, se utilizan dos pasos para *FindResource* y *LoadResource*, normalmente encontrará que se hace uso de estas funciones de la siguiente manera:

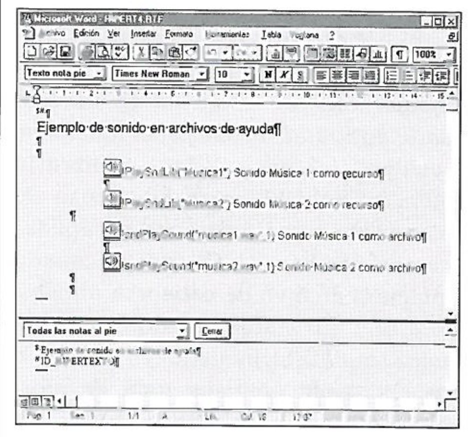


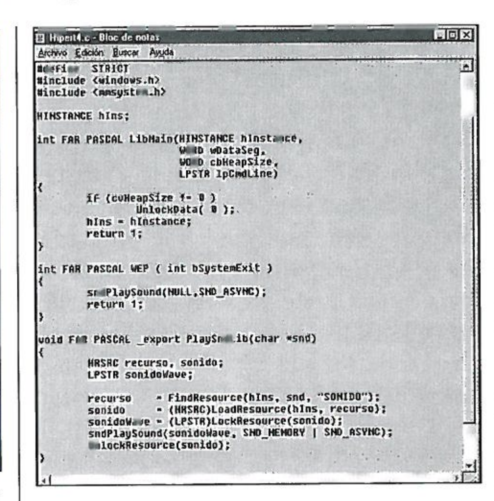
Figura 4. Archivo RTF.

FIGURA 2		
Carácter	**Tipo de dato**	**Tipo en Windows**
u	unsigned short	UINT, WORD, WPARAM
U	unsigned long	DWORD, RGBQUAD
i	signed short	BOOL
I	signed long	LONG, LPARAM, LRESULT
s	far char *	LPSTR, LPCSTR
v	void	Ninguno

Equivalencias con C de la macro RegisterRoutine, para el formato de parámetros.

FIGURA 3	
SND_SYNC	0
SND_ASYNC	1
SND_NODEFAULT	2
SND_MEMORY	4
SND_LOOP	8
SND_NOSTOP	10

Valores de los flags de la función sndPlaySound.



```
Hipert4.c - Bloc de notas
Archivo Edición Buscar Ayuda
#define STRICT
#include <windows.h>
#include <mmsystem.h>

HINSTANCE hIns;

int FAR PASCAL LibMain(HINSTANCE hInstance,
                       WORD wDataSeg,
                       WORD cbHeapSize,
                       LPSTR lpCmdLine)
{
        if (cbHeapSize != 0 )
                UnlockData( 0 );
        hIns = hInstance;
        return 1;
}

int FAR PASCAL WEP ( int bSystemExit )
{
        sndPlaySound(NULL,SND_ASYNC);
        return 1;
}

void FAR PASCAL _export PlaySndLib(char *snd)
{
        HRSRC recurso, sonido;
        LPSTR sonidoWave;

        recurso    = FindResource(hIns, snd, "SONIDO");
        sonido     = (HRSRC)LoadResource(hIns, recurso);
        sonidoWave = (LPSTR)LockResource(sonido);
        sndPlaySound(sonidoWave, SND_MEMORY | SND_ASYNC);
        UnlockResource(sonido);
}
```

Figura 5. Código fuente de la DLL.

LoadResource(hIns, FindResource(hIns, snd, "SONIDO"))

Aunque pudiera parecerlo, *LoadResource* no carga realmente el recuso en memoria, la función encargada de esto es *LockResource*, que bloquea y obtiene un puntero al recurso. Una vez se ha terminado de usar el recurso es necesario desbloquearlo con la función *UnlockResource*. La figura 5 muestra el código de la función *PlaySndLib*.

LA FUNCIÓN sndPlaySound

El API de Windows posee una librería completa de funciones para multimedia, una de éstas es la función *sndPlaySound*. Todas estas funciones residen en la librería MMSYSTEM.DLL, para hacer uso de alguna de estas funciones en aplicaciones escritas en C, será necesario incluir el archivo de cabecera *mmsystem*.h. El archivo de ayuda *win31mwh.hlp* incluido en las SDK de Microsoft, contiene toda la información a cerca de MMSYSTEM.

La función *sndPlaySound* es capaz de reproducir un archivo de sonido WAV, aunque con algunas limitaciones, esta función es más que suficiente para nuestros propósitos. El prototipo de *sndPlaySound* es el siguiente:

BOOL sndPlaySound(LPCSTR nombreSonido, UINT flags)

El primer parámetro es el nombre del sonido, si no encuentra el archivo de sonido, se ejecutará el sonido por defecto del sistema, si el valor es NULL, se detiene cualquier sonido que se esté ejecutando. El segundo parámetro especifica varias opciones para ejecutar el archivo de sonido WAV, pueden ser combinados con el operador OR (|), los valores posibles son los siguientes:

- *SND_ASYNC* ejecuta el sonido simultáneamente con otros procesos, será posible seguir trabajando mientras se ejecuta el sonido.

- *SND_LOOP* indica que el sonido debe repetirse indefinidamente. Si se emplea conjuntamente con *SND_SYNC*, no habrá posibilidad de parar el sonido.

- *SND_MEMORY* cambia el significado del primer parámetro, indica que éste es un puntero a memoria donde están los datos del sonido WAV. Como es el caso del ejemplo que acompaña a este artículo, donde lo sonidos están como recursos.

- *SND_NODEFAULT* especifica que si no encuentra el archivo de sonido, no ejecute el sonido por defecto del sistema.

- *SND_NOSTOP* especifica que el sonido no será interrumpido por la ejecución de otro sonido.

- *SND_SYNC* indica que la función no devuelva el control hasta que el archivo de sonido se termine de ejecutar.

En el ejemplo que acompaña a este artículo la función WEP llama a *sndPlaySound* con NULL como primer parámetro, la razón de esto es que es necesario terminar cualquier sonido que se esté ejecutado en memoria antes de descargar la DLL, al descargarse la DLL se libera la memoria, y si se estuviese ejecutando algún recurso WAV se produciría un error, este problema no existe para los sonido que se ejecuten desde archivo o los que no lleven el flag *SND_ASYNC*.

EL EJEMPLO QUE ACOMPAÑA ESTE ARTÍCULO

Para que el ejemplo funcione correctamente, será necesario disponer de una tarjeta de sonido compatible con Windows. Aunque el programa no detecta la presencia de una tarjeta de sonido, no se producirá ningún error en equipos que no dispongan de sonido, simplemente no se oirán los sonidos. Sería un buen ejercicio para todos los lectores de Sólo Programadores intentar implementar el código necesario para detectar si Windows tiene capacidad de sonido.

ANALIZADORES LÉXICOS

Daniel Navarro



A partir de la definición del sencillo lenguaje de programación Letra en el anterior capítulo de esta serie se iniciará la creación de un compilador para él , construyendo la que es la primera parte de todo compilador: el analizador léxico.

Un analizador léxico es la parte de un programa que gestiona la entrada de un fichero de texto escrito en un lenguaje determinado. Funcionalmente este analizador proporciona dos servicios o procedimientos al programa.

Inicialización_léxica(fichero_de_entrada)
Obtener_pieza()

Con el primero se puede indicar cuál es el fichero fuente que se va a tratar, y con el segundo se van obteniendo PIEZAS SINTÁCTICAS (o 'tokens') del fichero de entrada hasta que el analizador indique que ha llegado la pieza 'EOF' (o fin del fichero).

Por ejemplo, si se ha construido un analizador léxico para el lenguaje C, se tendrá definida la lista de piezas sintácticas del lenguaje (con un código asociado a cada una) que puede reconocer este analizador, algo no muy diferente a esto:

enum piezas { p_#, p_define, p_include, p_pragma, p_main, p_abrir_paréntesis, p_cerrar paréntesis, p_int, p_char, p_long, p_punto_y_coma, ... p_EOF }

Para tratar un programa en C, no se hará carácter a carácter, sino pieza a pieza, dejando que analizador léxico sea el encargado de 'trocear' la ristra de carácteres que conforma el fichero analizado en las correspondientes piezas. Primero se inicializará su analizador léxico con el fichero a tratar:

Inicialización_léxica("prueba.c");

Y tras ello tendremos una función *(Obtener_pieza())* que cada vez que sea invocada nos devolverá la información que necesitemos sobre la siguiente pieza leída del fichero.

No es muy complicado construir un analizador léxico, al menos no tanto como definir bien las piezas sintácticas del lenguaje que se va a analizar.

DEFINICIÓN DE LAS PIEZAS

Obviamente, para definir bien la lista de piezas de un lenguaje, se debe antes haber definido bien el lenguaje.

Para empezar con un ejemplo sencillo, se supondrá que se quiere construir un analizador léxico con el fin de analizar un lenguaje que sirva para configurar una aplicación desde un fichero de texto ASCII externo.

En dicho lenguaje, básicamente se pueden asignar constantes a diversas variables clave de la aplicación, veamos la estructura de un programa en este particular lenguaje con el siguiente ejemplo:

```
Inicio
  Usuario= "EC-00821-KXP";
  Tamaño_imagen= 640, 480;
  Color_fondo= 15;
  Color_tinta= 0;
  Ratio= 18.043;
  Salvapantallas= Off;
Fin
```

Las piezas sintácticas de este lenguaje podrían ser:

Las palabras reservadas (*Inicio, Fin, On, Off, Usuario, Tamaño_imagen, Color_fondo, Color_tinta, Ratio, Salvapantallas, ...*).

Los símbolos (igual (=), punto y coma (;), coma (,), ...)

Cuando se plantea el proceso de un fichero de texto siempre surgen los mismos problemas; los saltos de línea, los espacios, las tabulaciones, el final inesperado del fichero, reconocer las diversas partes, etc.

Más el resto de piezas que conforman un programa en dicho lenguaje pero no tienen una forma fija predeterminada, como son: literales (textos entre comillas), constantes enteras, constantes reales, ...

Con este último grupo de piezas es con las que se debe tener un mayor cuidado en definirlas correctamente, pues el analizador léxico debe determinar sin ningún tipo de conflicto:

a) Cuándo (en qué carácter del fichero) comienza dicha pieza.

b) Cuándo (en que carácter del fichero) termina.

c) Cómo y donde pasarnos información adicional sobre dicha pieza.

Se puede definir dichas piezas de la siguiente forma:- Literal; comienza cuando viene un carácter comilla doble (") y continúa mientras no venga otro carácter comilla doble.

- Constante entera; comienza cuando viene un carácter numérico (entre '0' y '9'), continúa mientras vengan más carácteres numéricos y finaliza con un carácter distinto a un punto (.).
- Constante real; comienza con un carácter numérico, continua mientras vengan más carácteres numéricos, tras los cuales vendrá un punto (de lo contrario se tratará de una constante entera) y tras éste uno o más carácteres numéricos.

Como alternativa al diseño de esta lista de piezas se podría haber optado por no definir constantes reales y definir el punto (.) como una pieza más del lenguaje, en cuyo caso debería ser la aplicación (y no el analizador léxico), la que debería reconocer dos piezas del tipo 'p_constante_entera' separadas por una pieza de tipo 'p_punto' como una constante real, pero esta solución no es la que se suele adoptar, ya que reconocer la pieza 'p_constante_real' es trabajo del analizador léxico.

Al analizador léxico le basta normalmente para indicar que pieza es la siguiente que se ha encontrado en el fichero con una variable en la que pueda indicar el código correspondiente a dicha pieza (por ejemplo en 'pieza_actual'). Es decir, si un analizador léxico de C se encuentra con la palabra '*main*' como siguiente pieza del fichero, le basta con indicar; '*pieza_actual = p_main*', pero si la pieza que viene a continuación es un literal, por ejemplo, no basta con indicar 'pieza_actual = p_literal', sino que además se debe pasar información sobre el contenido de dicho literal. Un poco más adelante se verá como se hará esto.

LAS REGLAS LÉXICAS

Básicamente las reglas léxicas son las que definen la forma de las piezas del lenguaje, como las que hemos definido en el punto anterior, aunque también se suelen definir otras reglas léxicas como pueden ser:

- Todas las palabras reservadas se aceptan en mayúsculas, en minúsculas o en cualquier combinación de ambas.
- Los literales deben comenzar y terminar en la misma línea, es decir que no pueden contener saltos de línea.
- Dos carácteres comilla doble ("") dentro de un literal se interpretan como un solo carácter comilla doble (") y se entiende que continúa dicho literal (ahora hasta que venga un, y solo un, carácter de dicho tipo).
- Las constantes numéricas no pueden tener más de 5 dígitos y su valor numérico correspondiente no puede ser superior a 32768.
- Etc.

Una vez definidas todas las reglas léxicas (y por consiguiente definidas todas las diferentes piezas del lenguaje) se puede comenzar a construir el analizador léxico, este irá informando de todas las piezas que conforman cada programa analizado, y si se encuentra algún carácter que no encaje exactamente con las reglas, nos informará de un error léxico.

INFORMACIÓN DE LOS ERRORES

En el ejemplo anterior, el analizador léxico informará de un error léxico cuando le pidamos la siguiente pieza y encuentre un literal que no termina en la línea en la que comenzó, una constante numérica demasiado grande (como 33456), una palabra que no coincide con ninguna de las palabras reservadas del lenguaje (como Salvapantallas), un carácter extraño que no pertenece a ninguna de las definiciones de símbolos del lenguaje (como #), etc.

Existen varias formas de tratar estos errores, una de ellas, la que se va a emplear en el analizador de Letra, consiste en añadir un nuevo tipo de pieza en la lista de piezas que denominaremos '*p_error*'.

INCIALIZACIÓN DEL ANALIZADOR

```
Inicialización_léxica( fichero_de_entrada ) {

  Abrimos ( fichero_de_entrada );
  Obtenemos longitud de ( fichero_de_entrada );
  Pedimos memoria para cargar el fichero;
  Cargamos el fichero en memoria;
  Cerramos ( fichero_de_entrada );
  Fijamos un puntero al inicio del fichero;
  Número_de_línea_actual = 1;

}
```

Listado 1.

Cuando el analizador se encuentre ante un error léxico, pasará esta parte del programa y a nuestra petición de obtener la siguiente pieza sintáctica nos responderá con que ha encontrado una *pieza_actual* de tipo *p_error*.

Será el programa que utilice el analizador léxico (el compilador este caso) el que decidirá de que forma va a tratar los errores.

CARACTERES NO SIGNIFICATIVOS

No todos carácteres del fichero analizado entran dentro de la definición de las piezas del lenguaje, sino que existen además los carácteres no significativos. Estos son ignorados por el analizador léxico, sin importar donde vengan ni cuántos vengan.

Cuando se define un lenguaje se debe dejar muy claro que carácteres (o combinaciones de ellos) pueden ser ignorados. Típicamente estos carácteres son los espacios en blanco, las tabulaciones y los saltos de línea. Pero además en casi todos los lenguajes se permite alguna forma de comentarios, estos permiten indicar al analizador léxico que queremos que ignore una parte determinada del programa. Los comentarios no son ninguna pieza sintáctica (pues no se informa de ellos al programa que esta usando el analizador léxico) pero debemos definir claramente cuando comienzan y cuando terminan, por ejemplo con reglas del tipo:

- Comentario; comienza con un carácter punto y coma (;) y termina al llegar un



MINISTERIO DE DEFENSA
DIRECCION GENERAL DE LA GUARDIA CIVIL
MINISTERIO DE CULTURA
MINISTERIO DE ECONOMIA Y HACIENDA
MINISTERIO DE INDUSTRIA, COMERCIO Y TURISMO
GENERALITAT DE CATALUNIA. DEP. SANITAT
SERVEI CATALA DE LA SALUT
BOLETIN OFICIAL DEL ESTADO



BANCO DE SANTANDER
INSTITUTO NCNAL. DE LA SEG. SOCIAL
MINISTERIO PARA LAS ADMINISTRACIONES PUBLICAS
FONDO DE GARANTIA DE DEPOSITOS
BANCO SANTANDER PUERTO RICO
BANCO ATLANTICO
PRICE WATERHOUSE
TOSHIBA



TELEFONICA
SANTANDER NATIONAL BANK
CORPORACION FINANCIERA HISPAMER
ANALISTAS FINANCIEROS INTERNACIONALES
SEGUROS OCASO
EUROSEGUROS BBV
SEGUROS ATHENA
FYCSA (ALCATEL)



CONSTRUCCIONES AERONAUTICAS (CASA)
PATENTES TALGO
SINTEL
AVIACO
AEROPUERTO DE CANARIAS
TRANSMEDITERRANEA
PUERTOS DEL ESTADO
GEC ALSTHOM
L OREAL



FASA RENAULT
RED DE CONCESIONARIOS RENAULT
REPSOL EXPLORACION
REFINERIA DE GIBRALTAR
PETROGAL
ONDA CERO RADIO
TELEMADRID



TELECINCO (GESTEVISION-TELECINCO)
UNIVERSIDAD AUTONOMA DE MADRID
UNIVERSIDAD CARLOS III DE MADRID
INSTITUTO DE EMPRESA
TELEFONICA SISTEMAS*
S.P. EDITORES*
OLIVETTI*



IBM*
SIEMENS NIXDORF*
DIGITAL*
BULL ESPAÑA*
FUJITSU*
INVESTRONICA*
SOFTWARE DE DIAGNOSTICO*



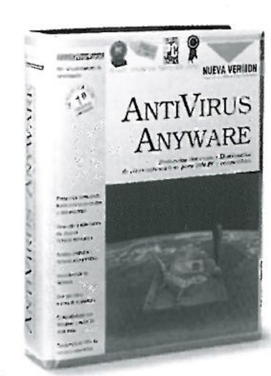
COMPUTER ASSOCIATES
G.P. INFORMATICA*
INFORMATICA EL CORTE INGLES*
ACTION*
GTI*
ALCATEL SISTEMAS DE INFORMACION*
INDAS



EL CORTE INGLES*
SOFTWARE DE ESPAÑA*
INFORMIX
ABBOTT CIENTIFICA
ALBILUX
ELIDA GIBBS
OXFORD UNIVERSITY PRESS

9 de cada 10 Ordenadores prefieren ANTIVIRUS ANYWARE.

El resultado está a la vista. O, ¿cree que tantas Empresas pueden estar equivocadas?. Tienen razones para no estarlo.



1. MAYOR PROTECCION.
Incorpora un Sistema de Protección con una ocupación mínima en RAM. Detecta el Virus antes de que pueda introducirse e impide el uso del fichero contaminado, avisándole a través de una ventana de alarma.

2. DETECCION INSTANTANEA.
De forma rápida y precisa, analiza cualquier unidad, directorio o fichero. Comprueba si su disco duro o disquetes contienen algún Virus. Chequea ficheros (incluso comprimidos), sector de arranque, tabla de particiones y unidades de red.

3. ELIMINACION DEFINITIVA.
Elimina los Virus allí donde se encuentren, sin dañar los ficheros.

4. SERVICIO DE ACTUALIZACION PERMANENTE.
Usted puede recibir las nuevas versiones cómodamente en su domicilio o capturarlas vía modem.

ANYWARE pone a su disposición el CLUB DE USUARIOS HELP VIRUS con una HOT LINE exclusiva para consultas, noticias, BBS, etc.

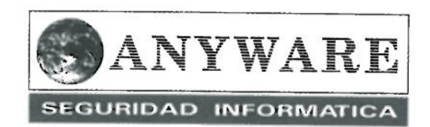









PREMIO PC MAGAZINE MEJOR UTILIDAD 1991 · RECOMENDACION PC WORLD MEJOR PRODUCTO ANTIVIRUS · RECOMENDACION PC MAGAZINE MEJOR PRODUCTO ANTIVIRUS · CATACOM 93 MEJOR ANTIVIRUS · PREMIO PC MAGAZINE MEJOR UTILIDAD 1993



Orense, 36 3º. 28020 MADRID. España.
Tel.: (91) 556 92 15. Fax.: (91) 556 14 04.

EMPRESAS E INSTITUCIONES CON LICENCIA CORPORATIVA DE ANTIVIRUS ANYWARE
(*)EMPRESAS DISTRIBUIDORAS DE ANTIVIRUS ANYWARE.

salto de línea o un fin de fichero. Todos los carácteres intermedios serán ignorados por el analizador léxico.

EL ANALIZADOR LÉXICO DE LETRA

Una vez vista la funcionalidad y uso general de un analizador léxico, se va a construir un analizador léxico para nuestro lenguaje. Retomando el diseño de Letra que se describió en el artículo del mes pasado, se describirán todas las diferentes piezas sintácticas que pueden formar un programa en letra.

No obstante, primero se definirán como carácteres no significativos los espacios, tabulaciones, saltos de línea y comentarios como los definidos en el punto anterior.

Primero, por un lado las palabras reservadas del lenguaje que son: { datos, cierto, falso, si, sino, fin, mientras, repetir, hasta, ir, hacer, volver, y, o, no }. A todas ellas les vamos a asignar códigos que denominaremos: *p_datos, p_falso,* etc. (estos códigos tendrán asociados números consecutivos; 0, 1, 2, ...)

Después otras palabras reservadas que son los nombres de las funciones especiales. En otros lenguajes, como C o Módula que tienen la posibilidad de cargar librerías los nombres de las funciones no son palabras reservadas del lenguaje, pero sí en nuestro caso (o en otros lenguajes como BASIC), pues estas son fijas y siempre las mismas. Estas son: *{ borrar_pantalla, escribir, leer_tecla, mover_cursor, color, aleatorio, esperar, leer_caracter, poner_caracter }*. Para distinguir estas piezas les asignaremos códigos con el mismo criterio: *p_borrar_pantalla, p_escribir,* etc.

Tanto las palabras reservadas del lenguaje como las de las funciones especiales se aceptarán indiferentemente en mayúsculas y/o minúsculas.

Después distinguiremos también como piezas del lenguaje los siguientes símbolos: distinto '<>', igual '=', mas '+', menos '-', multiplicar '*', dividir '/', módulo '%', abrir corchete '[', mayor o igual '>=', menor '<', mayor '>', menor o igual '<=', cerrar corchete ']', abrir paréntesis '(', cerrar paréntesis ')', dos puntos ':' y coma ','. A estos símbolos también se les asignarán sus códigos de pieza correspondientes (*p_distinto, p_igual, p_mas, ...*)

Además las palabras reservadas y símbolos se definirán las siguientes piezas sintácticas en letra:

Identificadores. Los identificadores serán los nombres de los datos (o variables del programa), estos serán también palabras, pero diferentes de las reservadas, y los definiremos como una cadena de carácteres que comienza con un carácter alfabético ('a' .. 'z'), continúa mientras vengan seguidos carácteres alfanuméricos o el carácter de subrayado bajo ('a'..'z', '0'..'9', '_') y además dicha cadena no debe coincidir con ninguna de las palabras reservadas. Su código de pieza será *p_identificador* para todos ellos.

Etiquetas. Servirán para identificar diversos puntos del programa y se definen como; una cadena de uno o más carácteres que comienza con un carácter punto '.' y continúa mientras vengan seguidos carácteres alfanuméricos o el carácter de subrayado bajo. Su código será *p_etiqueta.*

Constantes enteras. Serán los valores numéricos que aparezcan en el programa y se definen como una secuencia de carácteres que comienza con un carácter numérico ('0'..'9'), continúa mientras vengan seguidos carácteres del mismo tipo y además su valor numérico no puede ser superior a 32768. Su código será *p_entero.*

Literales. Serán textos entrecomillados, y se definen como una secuencia de carácteres que comienza con un carácter comilla doble y termina con otro carácter idéntico en la misma línea. También se aceptarán literales que empiecen y acaben con el carácter comilla simple (o apóstrofe). Su código será *p_literal.*

Fin de fichero. Lo definiremos como una pieza más del lenguaje, pues cuando llegue el analizador léxico indicará que ha llegado una pieza con el código *p_eof* (end of file).

Error. No es propiamente una pieza del lenguaje, pero el analizador léxico indicará que ha llegado la pieza *p_error* cuando detecte un error.

FUNCIONAMIENTO DEL ANALIZADOR.

Como se aclaró antes, un analizador léxico no es normalmente un programa en sí, sino una parte de otro programa más complejo, pero como ilustración de este artículo se puede encontrar en el disco de la revista el código de un analizador léxico de Letra (programado en C), este es un programa que recibe como entrada un programa cualquiera en Letra y produce como salida un listado con todas las piezas sintácticas que se han encontrado en dicho programa.

La estructura de un analizador léxico es la siguiente, primero tenemos una función de inicialización cuyo esquema básico se puede observar el listado 1 que se encarta de fijar los valores de las principales variables relacionadas con el fichero.

Tras la inicialización el programa que esté usando el analizador léxico retoma el control, tras ello podrá llamar a la función principal del analizador léxico: *Obtener_pieza()*, y cuando esté preparado para leer la siguiente pieza (en nuestro caso después de informar en el fichero de salida de cuál era la primera pieza) llamará a la función de nuevo, repitiendo el proceso hasta que se encuentre con la pieza *p_eof* que indica que se llegó al final del fichero de entrada. El esquema de la función para obtener la siguiente

OBTENER LA SIGUIENTE PIEZA

```
Obtener pieza() {

  Obtener_carácter();

  si (estamos en el final del fichero)
    pieza_actual=p_eof;
  si_no según que ( carácter_actual ) {

    (es alfabético): leer_una_palabra();
    (es numérico): leer_un_número();
    (es un punto): leer_una_etiqueta();
    (es comillas): leer_un_literal();

    (es '+'): pieza_actual=p_mas;
    (es '-'): pieza_actual=p_menos;
    (es '*'): pieza_actual=p_multiplicar;
    (...)

    (si no es ninguno): pieza_actual=p_error;
  }

}

Obtener_carácter() {

  mientras (estemos en un carácter no significativo
           y no se haya acabado el fichero)
  {
    Avanzar el carácter;
    si (era un salto de línea)
      número_de_linea_actual++;
    si (empieza un comentario)
      pasar hasta el próximo final de línea;
  }

}
```

Listado 2.

pieza se puede observar en el listado 2.

En dicha función lo primero será llamar a *Obtener_carácterl();* esta será la función encargada de saltar todos los carácteres no significativos que vengan seguidos (espacios, tabulaciones y saltos de línea) y los comentarios, al finalizar dicha función pueden haber sucedido una de estas dos cosas:

a) Estamos posicionados en un carácter significativo, este será el comienzo de una pieza sintáctica del lenguaje o un error en caso contrario.

b) Hemos llegado al final del fichero.

Si se está en el primero de estos dos casos, entonces según el carácter que venga se actuará diversas formas:

Viene un carácter alfabético; se llamará a una función que lee los siguientes carácteres alfanuméricos que vengan y después mira si la pieza en cuestión es una de las palabras reservadas, si coincide con alguna de ellas pondrá en *pieza_acutal* el código correspondiente a dicha palabra, en caso contrario *pieza_actual* será *p_identificador*, pues dicha secuencia será analizada como un identificador de dato (o en otras palabras, un nombre de variable).

Viene un dígito numérico, una función leerá el los demás dígitos que vengan consecutivos y calculará su valor numérico, si este es mayor que 32768 informará de una pieza *p_error*, en caso contrario de una pieza *p_entero*.

Viene un punto, se informa de una pieza de tipo *p_etiqueta* y se avanzan los carácteres alfanuméricos correspondientes a dicha etiqueta.

Viene una comilla (simple o doble), sí dicho literal acaba en la misma línea se informa de una pieza *p_literal*, en caso contrario se avanza hasta el final de la línea y se informa de una pieza de tipo *p_error*.

Para reconocer el resto de los símbolos se procede de la siguiente forma:

- Si hay varios símbolos que comiencen con el mismo carácter (como '<', '<=' y '<>') entonces se llama a una función que distinga entre ellos.
- En caso contrario: Si el símbolo tiene más de un carácter, se comprueba que los siguientes carácteres coincidan con la definición del símbolo (de lo contrario se informará nuevamente de una pieza *p_error*) y entonces se indicará su código en *pieza_actual*. Si el símbolo es de un solo carácter, informaremos directamente de su código de pieza.

En todos los casos, tras analizar una pieza siempre se hacen dos cosas; indicar su código en pieza_actual y avanzar los carácteres de dicha pieza que ya no se volverán a leer.

No es siempre necesario que hayan espacios separando las distintas piezas del programa, solo en los casos en los que sea necesario (por ejemplo es necesario separar una palabra reservada de un identificador, pues de lo contrario todo sería tomado como una palabra en conjunto), pues la función *Obtener_caracter*() avanza carácteres no significativos solo cuando los haya en la posición actualmente analizada del fichero.

Para facilitar la lectura, se suele añadir un retorno de carro al final del fichero, de este modo sólo se tendrá que comprobar si hemos llegado al final cuando se encuentren saltos de línea en el fichero. Además, en algunos casos se debe mirar algún carácter más allá de los reconocidos (por ejemplo para reconocer un símbolo mayor '>' tenemos que asegurarnos de que el siguiente carácter no es un carácter '=') y de este modo, al asegurar que al final del fichero siempre hay un retorno de carro, no se tiene que preguntar si se ha acabado el fichero cada vez que se requiere mirar el siguiente carácter.

INFORMACIÓN ADICIONAL

Cuando el analizador informa que ha encontrado una de las piezas que requiere otra información adicional para identificarla además del tipo de pieza (como p_identificador, p_etiqueta, p_entero o p_literal), debe indicarla en alguna parte. En el caso del analizador léxico de letra, esta información se indica en un vector (o cadena de carácteres, que en el ejemplo del disco se denomina nombre[]) y la información guardada según el tipo de pieza_actual analizada es la siguiente:

p_identificador: en el vector se guarda el identificador pasado a minúsculas.

p_etiqueta: se guarda el nombre de la etiqueta incluyendo el punto inicial y también pasado a minúsculas.

p_entero: se guarda el entero también como una cadena de carácteres (como sus correspondientes dígitos numéricos) pero asegurando ya que es un valor entero válido y no superior a 32768.

p_literal: se guarda en el vector una cadena con el contenido del literal, sin incluir ya los delimitadores (comillas simples o dobles).

Como se puede observar, en todos los casos, la información requerida para estas piezas es guardada como una cadena de carácteres, aunque esto no tiene por que ser así siempre.

CONSTRUYENDO OTROS ANALIZADORES LÉXICOS.

En el ejemplo presentado se simplifican algunas cosas, que probablemente se deban corregir si se programa una aplicación en serio, estas son las principales:

- Se supone que el fichero de entrada se puede cargar entero en memoria de una sola vez, si se quisiera tratar ficheros más grandes se tendría que implementar una carga por bloques.
- No se da información acerca de los diferentes errores léxicos encontrados (carácter extraño, constante entera demasiado grande, literal sin cerrar, ...). Hacer esto sería tan sencillo como tener una variable como *'tipo_error'* y cuando se produjera un error se indicaría en dicha variable un código según el tipo de error (además de responder que la pieza encontrada es de tipo *p_error*).
- Se calcula la línea, pero no columna en la que está localizada cada pieza en el fichero (esto es importante para poder dar información acerca de los errores).

Por lo demás, siguiendo el esquema básico del ejemplo aquí presentado, no será muy difícil construir otros analizadores léxicos, siempre que previamente se hayan definido bien tanto el lenguaje, como todas las piezas sintácticas a reconocer.

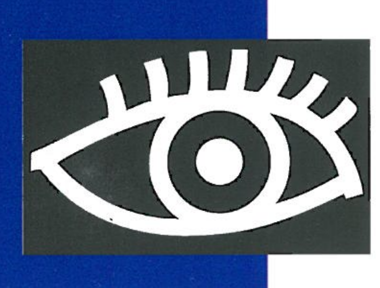
PRÓXIMO NÚMERO

Una vez tenemos construido este módulo del compilador de Letra, ya no se volverá a hablar de comentarios, ni líneas, ni espacios, etc. ni siquiera del fichero fuente, sólo se hablará de aquí en adelante a nivel de piezas sintácticas, tendremos una lista con todas las que componen el programa que vamos a analizar, cada una con la información necesaria.

Se pasará a continuación a estudiar el análisis sintáctico, tras esto se podrá ver ya el auténtico núcleo de un compilador en la fase de análisis: la tabla de símbolos.

EL FORMATO FLI/FLC

Agustín Guillén



Frente a los modernos formatos de animación, tales como el vídeo JPEG y los ficheros AVI, todavía permanece el "clásico" formato FLI/FLC. Debido a su estructura, su uso se limita principalmente a la infografía, pero como esta última continúa en auge, el futuro próximo de los ficheros FLIC está asegurado.

Hace ya bastantes años, Autodesk inventó el formato FLI. Fue diseñado para poder almacenar y reproducir las animaciones creadas mediante su programa Animator. Rápidamente se convirtió en el formato estándar para almacenar animaciones gráficas debido a su relativa sencillez y a la aparición de programas reproductores y conversores de libre distribución. En aquella época se estimó que animaciones realizadas en resoluciones mayores que el estándar MCGA (320 x 200 pixels) no se podrían reproducir a la velocidad adecuada, y como las primeras tarjetas SVGA eran totalmente incompatibles unas de otras, se optó por limitar el formato a resoluciones gráficas de 320 x 200 pixels. Claramente, como ocurre siempre que se decide algo en función de las prestaciones de los equipos, las limitaciones se quedaron anticuadas al cabo de unos años, por lo que Autodesk diseñó otro formato, o mejor dicho, amplió el formato FLI y creó el formato FLC, utilizado en su producto Animator Pro. En este último se eliminó la barrera de la resolución MCGA y se implementaron las nuevas resoluciones de que eran capaces los equipos más recientes: 640 x 480, 800 x 600, 1024 x 768 y 1280 x 1024, además, claro está, de la clásica 320 x 200. Por otro lado se incorporaron nuevos tipos de compresión y un control más amplio y sutil sobre los colores de la paleta de la animación.

Sus ventaja es que está orientado al almacenaje de las diferencias que se van produciendo entre distintos cuadros o frames dentro de una animación, por lo que sólo guarda los pixels que van cambiando. Esto le hace inmejorable en animaciones realizadas a base de bitmaps o figuras creadas por ordenador en las que el movimiento es suave y progresivo y en las que el fondo apenas cambia. El formato JPEG y la mayoría de las versiones del formato AVI están orientados a secuencias de vídeo o imágenes foto-realísticas por lo que cuando son utilizadas para animaciones infográficas, a menudo ocupan mucho más que la propia suma de todas las frames. Esto es reversible y si se utiliza el formato Flic para animaciones de vídeo, obtendremos unos ficheros enormes y unas prestaciones muy pobres en la velocidad de reproducción. Por otro lado, también carece de la posibilidad de incorporar sonido conjuntamente con las imágenes y se debe sincronizar manualmente si se desean reproducir sonidos. Por último, como sus algoritmos de compresión-descompresión son sencillos, está totalmente libre de requerimientos hardware, al contrario del formato JPEG y de la mayoría de los AVI, pues estos utilizan algoritmos tan complejos, que necesitan para su reproducción tarjetas especiales que incorporan potentes codificadores.

Los ficheros FLC se componen de una cabecera de 128 bytes, seguida de un *chunk* o bloque opcional a modo de prefijo y de uno o más de estos bloques conteniendo las distintas escenas de la animación (en toda la documentación existente sobre este formato, a cada bloque almacenado se le denomina *chunk*). Cada uno de estos bloques se suele componer de varios sub-bloques, dependientes de él.

El bloque de prefijo contiene datos y configuraciones propias del programa Animator Pro, tales como posiciones de los gráficos CEL sobreimpresos e informaciones acerca del autor, fechas, etc. Los bloques siguientes contienen los cuadros o secuencias propias de la animación, existiendo adicionalmente otra secuencia final que sirve de enlace opcional entre las secuencias última y primera de la animación, muy útil para realizar animaciones circulares y repetitivas. Cada bloque contiene información sobre pixels y/o sobre colores de la paleta.

Offset	Long	Descripción
0	4	Tamaño del bloque, incluyendo la cabecera y todos los sub-bloques contenidos en él.
4	2	Tipo de bloque. F100H para el bloque de prefijo y F1FAH para los bloques convencionales.
6	2	Número de sub-bloques contenidos en el bloque.
8	8	No utilizado, debe rellenarse con ceros.

Figura 2.

F1FAH para el resto de las secuencias. Salvo para el propio Animator Pro, no tiene demasiada utilidad el leer este bloque, pues la información contenida no es trascendente. Habitualmente se saltan sus datos, mediante su tamaño indicado

Offset	Long	Descripción
0	4	Tamaño del fichero, incluyendo la cabecera.
4	2	Identificador del fichero AF12H para FLC, AF11H para FLI.
6	2	Número de secuencias o frames.
8	2	Anchura de la pantalla (pixels). En los ficheros FLI, siempre será 320.
10	2	Altura de la pantalla (pixels). En los ficheros FLC, siempre será 200.
12	2	Bits por pixel (siempre será 8).
14	2	Flags. Se escribe 0003H después de terminarse de generar el fichero. Indica que el fichero se completó adecuadamente en su generación.
16	4	En los ficheros FLC, número de milisegundos de retardo entre cada secuencia de la animación. En los ficheros FLI este retardo está indicado en 1/70 partes de segundo y solo se compone de 2 bytes.
		A partir de este punto los datos son exclusivos a los ficheros FLC, ya que en el formato FLI, deben estar a 0.
20	2	No utilizado, debe rellenarse con ceros.
22	4	Fecha y hora de la creación del fichero (formato MS-DOS).
26	4	Número de serie de la copia del Animator Pro que creó el fichero. En otros casos su contenido es variable.
30	4	Fecha y hora de la última modificación en el fichero (formato MS-DOS).
34	4	Número de serie de la copia del Animator Pro que realizó la última modificación en el fichero. En otros casos su contenido es variable.
38	2	Aspect ratio en el eje X. Proporción en la coordenada X con la que la animación fue creada.
40	2	Aspect ratio en el eje Y.
42	38	No utilizado, debe rellenarse con ceros.
80	4	Offset o desplazamiento a la primera secuencia de la animación, desde el inicio del fichero.
84	4	Offset o desplazamiento a la segunda secuencia de la animación, desde el inicio del fichero. Es utilizado cuando se salta directamente desde la secuencia de enlace a la segunda, durante la reproducción.
88	40	No utilizado, debe rellenarse con ceros.

Figura 1.

El formato de la cabecera (128 bytes) se puede observar en la figura 1.

El formato del bloque inicial es idéntico al de los bloques convencionales, la única diferencia reside en el campo que indica el tipo de bloque. Este campo contiene el valor F100H para el bloque inicial o de prefijo y el valor

en su cabecera o mediante el offset de la cabecera principal que apunta directamente al primer bloque de la animación.

El formato de los bloques o *chunks* (16 bytes) se aprecia en la figura 2.

A continuación de cada cabecera de bloque, se encuentran los sub-bloques, tantos como se indiquen en dicha cabecera. Si el número de sub-bloques indicado es 0, quiere decir que no se han producido cambios en la animación (en pantalla) entre las dos últimas secuencias, pero aún así se debe realizar el retardo indicado en la cabecera del fichero.

El formato de los sub-bloques (longitud variable) es el de la figura 3.

Mediante el campo que señala el tipo de sub-bloque, se indica el método de compresión utilizado y de qué clase son los datos almacenados.

A continuación se van a detallar los distintos tipos de sub-bloques definidos, en el título de cada uno se muestra su código o valor, su nombre y su descripción abreviada:

4 - FLI_COLOR256 - Información de la paleta de 256 niveles

No disponible en los ficheros FLI. La información está organizada en paquetes y mediante la primera word del sub-bloque se indica su número. Cada paquete consta de la siguiente estructura:

- Número de colores a saltar.
- Número de colores a cambiar.
- Cadena de bytes definiendo los nuevos colores (3 bytes para cada color).

Con el siguiente ejemplo, se puede clarificar un poco esto:

Offset	Long	Descripción
0	4	Tamaño del sub-bloque, incluyendo esta cabecera.
4	2	Tipo de sub-bloque.
6	(Tamaño del sub-bloque indicado - 6)	Los datos sobre los pixels o los colores a cambiar, dependiendo del tipo de sub-bloque de que se trate.

Figura 3.

Los datos del sub-bloque
3
1,1,r,g,b
2,2,r,g,b,r,g,b
2,3,r,g,b,r,g,b,r,g,b
Significarían
Tres paquetes o definiciones.
Saltar un color y definir un sólo color (el color número 1)
Saltar dos colores y definir dos colores más (colores número 4 y 5)
Saltar otros dos y definir tres (colores 8, 9 y 10)

Si en el número de colores a cambiar aparece un 0, indicaría que se van a cambiar los 256 colores posibles. Los valores r, g y b (rojo, verde y azul en inglés) que definen un color permiten 256 niveles cada uno (entre 0 y 255), con lo que se obtiene una definición True Color o de color real.

7 - FLI_SS2 - Compresión delta orientada a words o palabras

No disponible en los ficheros FLI. Contiene las diferencias entre la secuencia anterior de la animación y la actual. Los datos están almacenados en líneas y estas a su vez en paquetes. La primera palabra del sub-bloque (después de la cabecera) informa del número de líneas contenidas en el sub-bloque. Cada línea puede comenzar con palabras especiales que realizan acciones adicionales. Para distinguir el contenido o el tipo de una palabra, se deben leer sus dos bits más altos, que informarán como en la figura 4.

Bit 15	Bit 14	Descripción
0	0	La palabra almacena el contador de los paquetes que vienen a continuación. Si el contador es 0, indica que solamente el último pixel de la línea ha cambiado.
1	0	El byte más bajo de la palabra debe colocarse en el último byte de la línea. Esto es útil en animaciones con un ancho de pixels impar. Un contador de paquete vendrá a continuación, en la siguiente palabra.
1	1	La palabra contiene un contador de salto de líneas. El número de líneas a saltar se obtiene mediante el valor absoluto (sin signo) de la palabra.

Figura 4.

El primer byte de cada paquete indica la posición X de inicio dentro de la línea actual (las línea se almacenan de abajo hacia arriba), mientras que el segundo byte señala el tipo de paquete. Si el tipo de paquete es positivo, entonces se copiaran tantas words como indique, directamente desde el paquete a la pantalla; pero si el tipo de paquete es negativo, entonces deberá repetirse la siguiente palabra del paquete tantas veces como marque el valor absoluto (sin signo) del tipo de paquete.

11 - FLI_COLOR - Información de la paleta de 64 niveles

Este sub-bloque es idéntico al FLI_COLOR256, salvo en que los valores que definen un color oscilan entre 0 y 63 (64 niveles).

12 - FLI_LC - Compresión delta orientada a bytes

Similar a los sub-bloques FLI_SS2, también contiene las diferencias entre dos secuencias consecutivas. Es el método habitual de compresión utilizado en los ficheros FLI, pero en los ficheros FLC, siempre se utiliza el método FLI_SS2.

La primera palabra contiene la posición de la primera línea a cambiar (indica el número de líneas sin cambios) y la segunda palabra señala las líneas contenidas en el sub-bloque. Los datos vienen a continuación y se diferencian con respecto al método FLI_SS2 en que los contadores y los movimientos de datos, son a base de bytes, no de palabras o words.

13 - FLI_BLACK - Secuencia completamente en negro

Este sub-bloque no contiene dato alguno e indica que la secuencia de animación siguiente debe ser creada con el color número 0 de la paleta, generalmente el color negro.

15 - FLI_BRUN - Compresión del tipo run lenght orientada a bytes

En este tipo de compresión se almacena la secuencia completa de la animación, por lo que es utilizada para guardar la primera secuencia o la pequeña imagen del sub-bloque FLI_PSTAMP. Su almacenaje discurre desde la parte superior de la pantalla hacia la inferior y está basado en líneas (tantas como altura de la animación).

Cada línea se basa en pequeños paquetes de compresión, y el número de paquetes se especifica en el primer byte de la línea. Esto es una reminiscencia del formato FLI y hoy en día debe ser ignorado, pues puede haber más de 255 paquetes en una línea. Para controlar el número de paquetes de una línea se tiene en cuenta la anchura de la animación y se van descomprimiendo pixels hasta alcanzarla. Los paquetes se componen de un byte que indica el tipo o el tamaño del paquete y de uno o varios más a continuación. Si el primer byte es positivo, significa que el segundo byte debe ser repetido tantas veces como indique, y si es negativo señala que se deben copiar directamente tantos bytes (del paquete a la pantalla) como su valor absoluto muestre.

16 - FLI_COPY - Sin compresión

Contiene una imagen de la secuencia sin compresión alguna. Suele utilizarse en los raros casos en los que con las compresiones habituales (FLI_SS2 o FLI_BRUN) se obtiene un tamaño mayor del de la imagen original. El tamaño de los datos (pixels) coincide con la anchura de la imagen multiplicada por su altura.

18 - FLI_PSTAMP - Imagen reducida

No disponible en los ficheros FLI. En este sub-bloque se almacena una pequeña versión de la primera secuencia de la animación. Esto será útil cuando se desea obtener un rápido ejemplo de lo contenido en un fichero FLC, por ejemplo en un selector de ficheros de anima-

```
color = 0
for rojo = 0 to 5
        for verde = 0 to 5
                for azul = 0 to 5
                        componente-rojo (color) = (rojo * 256) / 6
                        componente-verde (color) = (verde * 256) / 6
                        componente-azul (color) = (azul * 256) / 6
                        color = color + 1
                end
        end
end
```

Figura 5.

Offset	Long	Descripción
0	4	Tamaño del sub-bloque, incluyendo esta cabecera.
4	2	Tipo de sub-bloque (será siempre un 18).
6	2	Altura en pixels de la mini-imagen.
8	2	Anchura en pixels de la mini-imagen.
10	2	Tipo de conversión para los colores. Aparecerá siempre un 1, indicando el espacio cúbico de color de 6 x 6 x 6.
12	(Tamaño del sub-bloqueindicado - 12)	Datos de la imagen.

Figura 6.

Los datos están almacenados con los formatos de compresión siguientes:

15	FPS_BRUN	Compresión del tipo run lenght orientada a bytes.
16	FPS_COPY	Sin compresión.
18	FPS_XLAT256	Tabla de color cúbica (6 x 6 x 6).

ciones. Para aislar completamente a la imagen contenida de la paleta o paletas incluidas en las secuencias de la animación, está almacenada utilizando una definición de color cúbica de 6 x 6 x 6. Para generar el array o cubo se utilizaría la siguiente formula de la figura 5.

Para acceder a esta cadena de componentes de color, teniendo como base cualquier color rgb, sólo hay que aplicar la formula:

*((6 * rojo) / 256) * 36 + ((6 * verde) / 256) * 6 + ((6 * azul) / 256)*

con lo que obtendríamos un color o pixel final.

Los sub-bloques del tipo FLI_PSTAMP contienen una cabecera mayor que el resto de sub-bloques, tendría el aspecto de lo mostrado en la figura 6.

Los dos primeros son idénticos a los formatos FLI_BRUN y FLI_COPY y el tercer tipo contiene una tabla o lista de conversión de color de 256 valores, en vez de una lista de pixels. Esto se utiliza cuando las secuencias de animación son muy pequeñas y no vale la pena el almacenar una versión reducida. Para interpretar este formato (FPS_XLAT256) se lee cada pixels de la primera secuencia de la animación y se accede a la tabla de color cúbica para obtener un color o pixel final.

EJEMPLO DE PROGRAMACIÓN

Se puede observar que, aun no siendo muy complejo, el formato FLC dispone de una gran variedad de opciones y posibilidades, por lo que se ha desarrollado un completo programa para ilustrar su uso, manejo y compresión. Se trata de un reproductor de ficheros FLC (y consecuentemente también de ficheros FLI), desarrollado íntegramente en ensamblador. Sus prestaciones son notables a pesar de que la decodificación de nuevas secuencias se realiza directamente desde el disco, por lo que una mejora obvia sería la de cargar la animación completa en memoria y reproducirla luego leyéndola desde allí.

La estructura del diseño del programa es la siguiente:

Inicialización: Se incluyen las funciones habituales de un fichero COM, como son la liberación de la memoria no utilizada (en la ejecución de programas del tipo COM, toda la memoria queda bloqueada) y el control y lectura de los parámetros de entrada.

Lectura e interpretación de la cabecera: Se comprueba que se trata de un fichero FLI/FLC y se leen sus datos genéricos. En función de dichos datos se activa la resolución de vídeo adecuada y se almacena el retardo a realizar entre las distintas secuencias.

Lectura de cada secuencia o frame: Se comienza un bucle que sólo finalizará con la pulsación de una tecla o por medio de un error producido en la lectura de una secuencia. Dentro de este bucle se realizan las siguientes acciones:

- Prepara la VGA para la siguiente secuencia Next_Frame
- Lee la cabecera de la siguiente secuencia.
- Trata cada uno de los sub-bloques o chunks contenidos en la secuencia y para cada uno de ellos:
- Lee el sub-bloque del disco.
- Decodifica el sub-bloque según el tipo de que se trate. Identifica el sub-bloque y obra en consecuencia, aplicando el algoritmo de descompresión adecuado. Se puede observar que alguno de ellos es bastante complicado como la rutina decode_ss2 que contiene varios tipos de paquetes con tratamientos diversos.
- Realiza el retardo correspondiente. Para ello se reprograma el canal 2 del timer del PC para que pueda temporizar en intervalos de 1 milisegundo, con lo que solo resta ir decrementando un contador para esperar el tiempo especificado en la cabecera del fichero.
- Comprueba la pulsación de alguna tecla. En cuyo caso finaliza el programa, restaurando el modo de vídeo por defecto del sistema.

También se deben tener en cuenta las funciones para el manejo de la tarjeta gráfica mediante la norma VESA, incluidas en el mismo fuente. Si se desea adaptar el programa para que funcione con otra tarjeta gráfica, sólo se deben sustituir estas rutinas por otras similares de acuerdo a las características de la nueva tarjeta. Estas funciones gráficas están diseñadas para el volcado rápido de datos a la pantalla y están programadas a bastante bajo nivel, por lo que su seguimiento y compresión requiere cierto tiempo.

PROGRAMACIÓN DE LA PALETA VGA

Enrique de Alarcón

Saber manipular y programar la paleta de la VGA es imprescindible para todos los creadores de vídeo-juegos, ya que gracias a ello, se pueden crear efectos especiales muy difíciles de conseguir por otros medios. En este artículo aprenderemos todo lo relacionado con el tema y daremos a conocer las aplicaciones más usuales que tiene.

Prácticamente todos los vídeo-juegos de calidad comercial (e incluso no comercial) de la actualidad, presentan a lo largo de su desarrollo, más de una paleta de 256 colores de una VGA normal, a si mismo también presentan efectos de cambios de iluminación tanto crecientes como decrecientes y efectos de movimiento a base de realizar rotaciones de color. Todo ello y más, se realiza programando únicamente la paleta VGA y cambiando los atributos de sus colores.

En este artículo explicaremos como está compuesta la paleta de la VGA, como funcionan los colores , como se programa y como se pueden realizar los efectos especiales anteriormente mencionados. Con estos conocimientos, el programador podrá usar todo ello en sus aplicaciones e incluso crear nuevos efectos no explicados, lo cual dará más creatividad a sus aplicaciones y presentaciones.

FUNCIONAMIENTO DE LOS COLORES EN UNA VGA

La paleta de una VGA posee 256 colores diferentes que se pueden usar simultáneamente en pantalla, y cada uno de estos colores, se define en la paleta con tres bytes. Cada byte corresponde al rojo, verde y azul respectivamente, y cada componente de color puede tener un valor de intensidad del 0 al 63. Por ejemplo, si queremos que el color 0 de la paleta sea un rojo puro y fuerte, sólo tenemos que poner a 63 el atributo de rojo y a 0 los dos restantes. Dado que son 3 bytes por color y que cada uno tiene 64 combinaciones (0-63), podemos deducir que podemos crear 262.144 colores diferentes (64^3, 6*3bits = 18bits) aun que sólo podemos mostrar 256 simultáneamente.

Cada vez que se dibuja un pixel en pantalla, el DAC examina el valor del pixel, que funciona como un puntero dentro de la paleta de color (con valor entre 0 y 255), extrae sus atributos y los convierte a valores analógicos comprendidos entre 0.0 y 0.7V (que sería aproximadamente multiplicar cada ATRIBUTO por 0.011V) y que son los valores reales que usa el haz de electrones para proyectar en pantalla.

PROGRAMACIÓN DE LA PALETA

Programar la paleta de la VGA se puede realizar tanto escribiendo directamente en los puertos del DAC (in/out), como mediante servicios de la BIOS, aun que este último método resulta tan lento que es mejor considerar que no existe.

Para programar los atributos de un color, sólo tenemos que escribir el número de color en el puerto 03C8h y escribir a continuación los tres componentes en el puerto 03C9h, en el orden de rojo, verde y azul (RGB). El código para programar una paleta entera que tengamos en memoria es sólo realizar un bucle de 256 iteraciones y que escriba los tres componentes (atributos) de un color en cada pasada. El código completo aparece en la figura 1.

También podemos extraer la paleta actualmente programada en la VGA realizando una operación muy similar e inversa. Para extraer por ejemplo los atributos del color 10, sólo tenemos que escribir el número de color que queremos leer en el puerto 03C7h y extraer

```
push ds
cld
sub si,si
mov ax,BUFFER_PALETA0
mov ds,ax
mov dx,03C8h
mov cx,256
 BUCLE1:
   mov al,cl
   out dx,al
   inc dx
   lodsb ; = mov al,ds:si / inc si
   out dx,al
   lodsb
   out dx,al
   lodsb
   out dx,al
   dec dx
 loop BUCLE1
pop ds
```

Figura 1.

los tres componentes leyendo tres veces seguidas el puerto 03C9h, de donde obtendremos el rojo, verde y azul respectivamente.

El código para extraer una paleta entera es el de la figura 2.

MAPA DE LOS PUERTOS DEL DAC USADOS

3C7h: Color del que leer los atributos de color.

3C8h: Color del que modificar los atributos.

3C9h: Escritura/lectura atributos de color.

Otros puertos:

3DAh: Estado Haz electrones (bit 3).

```
cld
sub di,di
mov ax,BUFFER_PALETA0
mov es,ax
mov dx,03C7h
mov cx,256
 BUCLE1:
   mov al,cl
   out dx,al ; Color del que extrae.
   add dx,2
   in al,dx  ; Atributo 0 del color CX
   stosb ; es:di
   in al,dx  ; Atributo 1 del color CX.
   stosb
   in al,dx  ; Atributo 2 del color CX.
   stosb
 sub dx,2
loop BUCLE1
```

Figura 2.

EFECTOS ESPECIALES

Para realizar la mayoría de efectos de paleta, y para dar las explicaciones del artículo, partiremos de la base que poseemos dos buffers de memoria donde tenemos capacidad para almacenar dos paletas. En la primera, almacenaremos la paleta de uso actual, donde iremos almacenando la paleta tras la ultima modificación realizada en ella, y en la segunda guardaremos la paleta original o la segunda paleta a como complemento de cálculos en algunos efectos, los buffers se llamarán BUFFER_PALETA0 y BUFFER_PALETA1 respectivamente.

Recordaremos que cada buffer son 256*3bytes, o sea, 768bytes.

EFECTO DE DESAPARICIÓN

Primero explicaremos el efecto que más se ha visto hasta ahora en todos los videojuegos y el cual aparece como unión entre pantallas estáticas de presentación, entre fases y otros...

El efecto es muy fácil de realizar y consiste en hacer que la imagen del monitor desaparezca a base de decrementar su intensidad de color. Para ello, sólo tenemos que reprogramar 64 veces una paleta en la VGA decrementando cada vez en 1 los atributos de los 256 colores que forman la paleta.

El efecto es fácil de entender, pues si cada atributo es la intensidad de brillo de un componente primario, el decrementar hacia 0 todos los atributos de la paleta es lo mismo que decrementar 1/64 de intensidad en todos los colores que forman la imagen (o sea, decrementar 1/64 de la intensidad de la imagen del monitor).

El código a realizar es decrementar y reprogramar 64 veces todos los atributos de la paleta en la VGA:

El código completo se encuentra en la figura 3.

EFECTO DE APARICIÓN

Para realizar este efecto, debemos seguir prácticamente los mismos pasos que en el anterior. Debemos empezar con la paleta 0 completamente a 0 (todos los componentes primarios con valor 0) y almacenar en el segundo buffer la paleta hasta la que queremos llegar cuando acabe el proceso de aparición. Una vez inicializados ambos buffers, sólo debemos realizar 64 iteraciones reprogramando e incrementando cada vez todos los atributos de color de la paleta 0 que no hayan llegado a los valores correspondientes de la segunda. Al acabar el bucle, habremos incrementado la intensidad de todos los atributos de la paleta 0 hasta los mismos valores correspondientes de la paleta 1 en cada color, dando así el efecto de que ha aparecido la pantalla. El efecto de ello quedaría como aparece en la figura 4.

```
mov ax,BUFFER_PALETA0
mov es,ax
mov cx,64
DECREMENTA_IMAGEN:
  push cx
  sub si,si
  mov cx,768
   DECR_PALETAB1:
     cmp byte ptr es:[si],0
     je DC_PC1
      dec byte ptr es:[si]
    DC_PC1:
   inc si
  loop DECR_PALETAB1

  sub si,si
  mov dx,03C8h
  mov cx,256
   BUCLE1:
    mov al,cl
    out dx,al
    inc dx
    mov al,es:si
    out dx,al
    inc si
    mov al,es:si
    out dx,al
    inc si
    mov al,es:si
    out dx,al
    inc si
    dec dx
  loop BUCLE1
 ; Aqui iria el código de
 ; de retardo temporizado
 pop cx
loop DECREMENTA_IMAGEN
```

Figura 3.

EFECTO DE TRANSFORMACIÓN

Crear un efecto de transformación de una paleta a otra, es muy fácil de realizar . Para ello sólo tenemos que instalar la paleta de origen en el buf-

```
push ds
mov ax,BUFFER_PALETA1
mov es,ax
mov ax,BUFFER_PALETA0
mov ds,ax
mov cx,64
APARECE_IMAGEN:
push cx
 sub si,si
  mov cx,768
  INCR_PALETAB1:
     mov bl,es:[si]
     mov al,ds:[si]
     cmp al,bl
     jae INCR_PALCONT
         inc byte ptr ds:[si]
    INCR_PALCONT:
    inc si
  loop INCR_PALETAB1

 sub si,si
 mov dx,03C8h
 mov cx,256
  BUCLE1:
     mov al,cl
     out dx,al
     inc dx
     lodsb
     out dx,al
     lodsb
     out dx,al
     lodsb
     out dx,al
     dec dx
  loop BUCLE1
pop cx
loop APARECE_IMAGEN
pop ds
```

Figura 4.

fer 0 y la paleta destino hasta la cual queremos llegar en el buffer 1.

Una vez tenemos esto, sólo es cuestión de realizar 64 iteraciones acercando en 1 los valores de todos los atributos de la primera paleta hacia los valores contenidos en la segunda. Para ello debemos comprobar en cada uno de los 768 atributos de la paleta si su equivalente en la paleta del buffer 1 es mayor, menor o igual. Si el equivalente es mayor, le sumaremos 1, si es menor le restaremos 1 y si es igual, no lo modificaremos. El código a usar antes de cada una de las 64 actualizaciones de la paleta en la figura 5.

EFECTO DE FALSO MOVIMIENTO

Si realizamos un decorado que contenga, por ejemplo, una cascada, sólo tenemos que usar una pequeña gama de colores de la paleta exclusivos para ella, y usar los colores en el gráfico de manera que los colores se sucedan en orden literal en la dirección deseada (en el ejemplo, hacia abajo, para hacer que le agua "caiga"). Después, con el dibujo en pantalla, sólo tenemos que desplazar hacia la derecha o hacia la izquierda los atributos de los colores a mover dentro de la paleta usada e ir reactualizando la paleta en la VGA. De esta forma, los colores en rotación, darán la sensación de movimiento de agua, en este caso como una corriente hacia abajo. Este efecto es también facial de programar, pero obliga a dibujar de forma especial el gráfico. También debemos recordar que el desplazamiento de atributos dentro de la paleta, tanto si es hacia la izquierda como hacia la derecha, será en bloques de 3 atributos y que los atributos del último color que interviene en la rotación deben ser colocados en el lugar del primer color.

EFECTO MONOCROMO

Convertir un color o toda una paleta a su equivalente monocromo es muy fácil, y se puede realizar también de dos maneras, llamando un servicio de la BIOS, o calculando factores a partir de los atributos de color. Para realizarlo mediante un servicio BIOS, sólo tenemos que ejecutar el siguiente código, y toda la paleta se convertirá en lo que se aprecia en la figura 6.

En este código, la primera interrupción es para habilitar la adición de escalas de grises. La segunda llamada, es para definir la escala de gris, indicando en BX el primer atributo de color a calcular y en CX el número de registros de color a partir del indicado que se modificarán.

Programar la paleta se puede realizar tanto directamente por puertos como mediante la BIOS

```
sub si,si
mov ax,BUFFER_PALETA1
mov es,ax
mov ax,BUFFER_PALETA0
mov ds,ax
 mov cx,768
 TRANS_PALETAB1:
    mov bl,es:[si]
    mov al,ds:[si]
    cmp al,bl
    jb TRANS_B1_INC
    je TRANS_B1_CONT
      dec byte ptr ds: [si]
      jmp short TRANS_B1_CONT
    TRANS_B1_INC:
    inc byte ptr ds:[si]
    TRANS_B1_CONT:
   inc si
 loop TRANS_PALETAB1
```

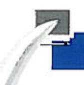
Figura 5.

Si queremos realizar nosotros los cálculos directamente, tenemos que saber las reglas de transformación, que son muy sencillas, debemos dejar de cada color un 30% del atributo de rojo, 59% del verde y un 11% del azul. Así, por ejemplo, un color que tenga atributos de R30,V50,A33, el resultado sería: Rojo=(30*30/100), Verde=(59*50/100) y Azul=(11*33/100).

Queda a elección del programador si que método quiere usar, que en realidad no es muy difícil de decidir, es sólo cuestión de velocidad y de si tenemos que trabajar sobre una paleta ya instalada en la VGA o si tenemos que aplicarlo sobre una paleta de memoria, ya que el servicio de la BIOS, funciona a partir de los atributos de la paleta activa en la VGA y no sobre una paleta de memoria.

IDEAS PARA OTROS EFECTOS

Un efecto más complicado, ya usado en algún que otro vídeo-juego, es la llamada <iluminación ambiental>. Esto consiste por ejemplo, en hacer que cuando un personaje lleva




```
mov ah,12h
mov al,00h
mov bl,33h
int 10h
mov ah,10h
mov al,1bh
mov bx,0000h
mov cx,0fffh
int 10h
```

Figura 6.

una luz a un lugar, los colores usados en el decorado, se incrementen de intensidad (incrementar valores de sus atributos), haciendo así el efecto de que realmente iluminamos el lugar. Un ejemplo de esto se puede ver en la última aventura gráfica de Indiana Jones, que se desarrolla en la Atlántida, donde hay un nivel donde indiana pasa por un pasadizo oscuro, y a medida que avanza, la zona donde está se va iluminando como si de un sistema de luz artificial se tratara. Esto consistía en dibujar las paredes y techos con colores iguales pero correspondientes a secciones diferentes de la paleta. El programa sólo tenía que cambiar la iluminación de los colores correspondientes a la zona donde se encontraba el protagonista.

Los cambios de intensidad de luz, pueden sernos de utilidad en muchas situaciones, ir cambiando el brillo de las estrellas en un simulador espacial, hacer efectos de disparos, efectos de electricidad, de niebla, etc...

VELOCIDAD CONSTANTE

Cada uno de los efectos que hemos mencionado y que deben realizarse en 64 (o las que queramos) pasadas, se deben realizar a una velocidad constante, pues si lo realizamos a la velocidad de proceso de la CPU, ni lo veremos. Para ello, tenemos dos opciones a incluir en los bucles de 64 iteraciones de cada efecto, o sincronizarlo con el temporizador, programando este al número de Hz que queramos (y el cual ya explicamos como programarlo en el numero 11, pagina 71), o sincronizarlo con el refresco de pantalla de la VGA, que funciona a una frecuencia constante (unos 60Hz normalmente) y que es sólo reprogramar la paleta cada vez que el haz de electrones del monitor está retrocediendo para volver a actualizar la imagen. Para ello, simplemente tenemos que leer el puerto 3DAh de la VGA y esperar a que el tercer bit cambie de valor pasando de ser 0 a 1, lo cual indica que el haz de electrones está retrocediendo. Como la VGA va a 60Hz y tenemos que realizar 64 pasadas, tardará algo más de un segundo en completar el efecto. Con este segundo sistema, evitamos también el llamado <efecto nieve> que consiste en ver puntitos de colores en pantalla momentáneamente debido a que cambiamos atributos de color al mismo tiempo que se están dibujando pixels en pantalla usando dichos colores, y lo cual resulta muy molesto a efectos visuales.

Este sistema de sincronismo se usa también en los scrolls de pantalla que son muy rápidos, haciendo así que no se escriban imágenes en el monitor mientras este todavía está actualizando la imagen anterior.

COMENTARIOS DE INTERÉS

Los tres primeros efectos, se han explicado de la forma más sencilla, pero podían haberse hecho también usando incrementos proporcionales de intensidades para hacer que todos los atributos adquieran los valores finales al mismo tiempo, para ello, tendríamos que coger todos los diferenciales de atributos (Estado_Final-Estado_Inicial) y dividirlos entre el número de pasadas que queremos realizar, de esta forma, todos los atributos, llegarían a su estado final en la misma iteración (la última). El problema es que de esta manera tenemos que usar decimales y los algoritmos se complican mucho. A elección del programador está la manera de realizarlo.

SOBRE LO EXPLICADO

Todo lo explicado sobre la paleta gráfica se basa en modos de 8bits de color (como el modo 13h, de 320*200 y 256 colores). En los modos SVGA de 32767, 65535 y 16'7 millones de colores, el control de los atributos de color, funciona de forma diferente pese a que los principios sean iguales o muy parecidos.

Por ejemplo, en el modo 16'7 millones de colores, cada color tiene también 3 atributos primarios, pero en vez de 64 niveles de intensidad, posee cada uno 256 y en vez de haber una paleta, cada pixel de pantalla se define directamente mediante los 3 bytes de atributos primarios, por eso se llama modo gráfico de 24bits (3bytes). Por eso, por ejemplo, para realizar el efecto de desaparición, no habría ninguna paleta que programar, si no decrementar los atributos de color directamente de la memoria de vídeo, lo cual resulta mucho más lento en cuanto a velocidad de proceso. En un caso de 1024*768pixels a 16'7milllones de colores, realizar una desaparición completa de imagen tono por tono, sería realizar 1024*768*3*64 operaciones de resta, que nos da 150.994.944, 151 millones de restas sin contar un número doble de accesos a memoria de vídeo, que precisamente es una memoria lenta de acceso en las tarjetas de vídeo normales.

En los modos SVGA de más de 256 colores, el control de atributos RGB, funciona de forma diferente

RESUMEN

Primero hemos explicado la causa por la cuál es útil saber programar la paleta de colore de la VGA, después hemos explicado los detalles técnicos de como funcionan los colores, sus atributos y limitaciones, para después, explicar los 4 efectos más aplicados en la paleta usados por todos los programadores. También comentamos alternativas a los ejemplos explicados, y otros modos de color que tienen las SVGA no compatibles en cuanto a programación con lo explicado.

CONCURRENCIA EN APLICACIONES WINDOWS

Francisco Otero

Son conocidas las limitaciones de Windows para ejecutar varias aplicaciones a la vez. Funciona aceptablemente bien cuando se trata de ejecutar varias tareas DOS a la vez, o tareas DOS y Windows simultáneamente, pero no permite la ejecución simultánea de varias tareas Windows.

Windows sólo permite concurrencia entre aplicaciones Windows,este comportamiento es debido a la capacidad de los procesadores 386 y posteriores de trabajar en modo virtual 86 (V86). El modo V86 es una emulación del procesador 8086, pero más rápido y con registros de 32 bits, corriendo en modo real. En un 386 se pueden ejecutar múltiples aplicaciones en modo V86, lo que equivale a múltiples emulaciones del modo real corriendo en multitarea entre sí (o con otras aplicaciones Windows), aprovechado las ventajas de la memoria virtual y la protección de código, así como de otras características de los 386/486.

Esta forma de multitarea se conoce con el nombre de multitarea preemtiva, cuando el procesador asigna un tiempo finito de CPU a cada tarea y conmuta de una a otra de forma transparente al usuario. Windows permite este tipo de multitarea entre aplicaciones DOS o entre aplicaciones DOS y Windows. Sin embargo, Windows sólo soporta multitarea no-preemtiva entre aplicaciones Windows, es decir, permite conmutar de una tarea a otra cuando la aplicación principal se encuentra esperando los mensajes que le envía el sistema, por ejemplo, esperando las entradas del teclado o del ratón, pero si una aplicación está realizando una operación no devolverá el control al sistema hasta que termine.

Si la duración de la tarea es pequeña, esta situación carece de importancia, pero si la duración del proceso es importante, supongamos un programa de render o un proceso de compilación, Windows, en muchos casos, no permitirá ejecutar o pasar a otra aplicación, y es posible que se tenga el ordenador ocupado durante minutos sin que se puedan realizar otras tareas.

EL PROGRAMADOR DEBE DE SUPLIR LAS CARENCIAS EN MULTITAREA PREEMTIVA DE WINDOWS

Para sacar el máximo rendimiento a un entorno multitarea se requiere que todas las aplicaciones compartan los distintos recursos hardware (memoria, cpu, impresora,...) del ordenador de forma racional. Es necesario tener la posibilidad de poder conmutar con toda facilidad de una aplicación a otra, y que, cuando una aplicación pasa a segundo plano trabaje con toda normalidad, al menos durante los intervalos en los que la aplicación principal no ocupa el procesador con cálculos intensivos. La mayoría de los programas comerciales Windows logran esta concurrencia, pero no gracias a la "bondad" del sistema, sino que se logra con un esfuerzo de programación por parte del fabricante.

Se necesitan técnicas de programación especiales para lograr que cualquier proceso se ejecute concurrentemente con el resto de aplicaciones Windows que puedan estar activas, y evitar así que se pueda colapsar el sistema cuando una ocupa la CPU durante minutos. Hay que tener en cuenta que uno de los principios, para la correcta programación en un sistema multitarea, es que no se deben ocupar todos los recursos durante un periodo de tiempo muy largo.

```
MSG msg;

while ( GetMessage( &msg, NULL, 0 ,0) )
    {TranslateMessage(&msg);
DispatchMessage(&msg);
    }
```

Figura 1. Bucle de mensajes básico.

SE DEBEN DE DETECTAR LOS INTERVALOS DE INACTIVIDAD DE WINDOWS PARA EJECUTAR EL PROCESO

Como todos los programadores de Windows que conocen la programación en DOS saben, la filosofía de programación en estos dos sistemas es bastante distinta. Los programadores DOS están acostumbrados hacer llamadas al sistema para hacer algo (salida por impresora, abrir un fichero, leer el teclado, ...), pero Windows es diferente.

Un programa Windows hace llamadas al sistema, pero a su vez, Windows hace llamadas al programa en ejecución (al crear la ventana, al pulsar una tecla o el ratón, al pintar una región de la pantalla, ...). El núcleo de Windows mantiene una cola de mensajes para cada programa que se está ejecutando en el sistema. Cada evento que ocurre, un movimiento del ratón, una ventana que queda al descubierto, una pulsación de teclado, el cambio de fecha del sistema..., se traduce en un mensaje que se coloca en la "cola de mensajes". El método ideado para implementar la concurrencia se basa en esa característica de Windows de ser un sistema gestionado mediante mensajes.

Todos los programas Windows tienen que procesar esa cola de mensajes, es un trozo de código que se conoce como el "bucle de mensajes". En la figura 1 se encuentra el bucle de mensajes escrito en lenguaje C.

La cola de mensajes permite saber lo saturado que está el sistema. Si la cola de mensajes está llena, Windows estará ocupado enviando mensajes a las aplicaciones correspondientes. Mensajes que a su vez tienen que ser procesados en los propios programas. Sin embargo, habrá muchas ocasiones en las que la cola de mensajes esté vacía, y Windows estará perdiendo el tiempo en algún bucle, esperando que ocurra algún evento que procesar. Se puede detectar esta situación, y aprovechar que no hay mensajes pendientes para obtener el control y hacer lo que queramos, devolviendo el control de nuevo al sistema cuando se añada un mensaje a alguna cola de mensajes.

El bucle de mensajes de la figura 2 permite aprovechar los ratos libres de Windows para la aplicación y luego devolver el control cuando el sistema se lo pida. La Diferencia más importe en la sustitución de la función *GetMessage* por *PeekMessage*.

Para sacar el máximo rendimiento a un entorno multitarea se requiere que todas las aplicaciones compartan los recursos hardware

GetMessage recupera el primer mensaje de la cola y lo borra, almacenándolo en una estructura de tipo *MSG*. Sin embargo, *PeekMessage* inspecciona la cola de mensajes e indica si existen mensajes que procesar.

Así el nuevo código del bucle de mensajes utiliza la función *PeekMessage* en vez de *GetMessage*, lo que permite que un programa tome el control cuando los demás programas hayan procesado sus mensajes y lo devuelve cuando los otros empiezan a trabajar.

Se inspecciona el valor devuelto por *PeekMessage*, si devuelve cero, la cola esta vacía y Windows inactivo, por lo que aquí se colocaría el código del proceso "background". Si devuelve algo distinto de cero se procesa el mensaje como de costumbre, tomando las otras aplicaciones el control.

```
MSG msg;

while (TRUE)
if ( PeekMessage (&msg, NULL, 0, 0,
PM_REMOVE) )
    {       if (msg.message == WM_QUIT) break;
            TranslateMessage(&msg);
            DispatchMessage(&msg);
    }
                        else
                                Mi_funcion( );  // Llamada
a la función que se ejecutará concurrentemente
```

Figura 2. Bucle de mensajes modificado para permitir concurrencia.

SI LA APLICACIÓN PUEDE PERDER EL CONTROL, ES NECESARIO TENER CUIDADO AL PROCESAR WM_PAINT

El método es bastante sencillo, sólo un par de aclaraciones. La primera es que la llamada a la función *PeekMessage* se realiza con un valor NULL en el segundo parámetro (el que indica la ventana a examinar). Esto es para que PeekMessage inspeccione en todas las colas de mensajes, no sólo en la de una determinada aplicación. El segundo comentario sobre el nuevo bucle de mensajes es que hay que procesar el mensaje *WM_QUIT* de forma especial. Esto es debido a que *GetMessage* devuelve cero si procesa un mensaje *WM_QUIT*, sin embargo *PeekMessage* no devuelve cero.

También se debe recordar que tanto *GetMessage* como *PeekMessage* no eliminan el mensaje *WM_PAINT* de la cola, por lo que un bucle para vaciar la cola de mensajes como el siguiente bloquearía el ordenador (en caso de existir algún *WM_PAINT*).

While (PeekMessage (&msg, NULL, 0,0, PM_REMOVE));

En el disco que se incluye con la revista se encuentra un programa de ejemplo que ilustra perfectamente esta técnica. Se trata de una versión para Windows de los clásicos ojos de X-Windows.




```
int c;
MSG msg;

for ( c = 0; c < 1000; c++ ) // se repite 1000 veces
  {// Aquí irá el código del bucle
        if  ( PeekMessage( &msg, NULL, 0, 0,
PM_REMOVE ) )
        SendMessage( msg.hwnd, msg.message,
msg.wParam, msg.lParam );
            }
```

Figura 3. Ejemplo de concurrencia en bucles.

Un último comentario sobre la concurrencia de procesos es que todas estas técnicas de programación de poco (o nada) sirven cuando un programa perfectamente programado para soportar concurrencia se ejecuta a la vez con otro que no las utiliza, y que mantiene ocupado durante cinco minutos al procesador en un bucle. No existe ninguna técnica software para obtener el control. De lo que sí se puede estar seguro es que, utilizando estas técnicas, una vez que acabe el proceso principal, el otro se activará y continuará su ejecución en segundo plano.

ES IMPORTANTE MANTENER LA CONCURRENCIA DEL CÓDIGO EN LOS BUCLES

El método explicado hasta ahora, es perfectamente válido para muchos programas, pero habrá otros que por su gran complejidad lógica, no puedan transformarse en aplicaciones Windows que cooperen con las demás cambiando únicamente el bucle de mensajes.

Supongamos que se quiere diseñar una aplicación que necesita recorrer un fichero para buscar una determinada información, u ordenar una matriz de grandes dimensiones, o cualquier algoritmo que consuma bastante tiempo dentro de bucles. Con la técnica vista hasta ahora, de modificación del bucle de mensajes, resultaría complicado implementar algoritmos concurrentes con funciones recursivas o con bucles anidados. La solución ideal en estos casos no es modificar el bucle de mensajes. Nos bastaría con incorporar, en los bucles que consumen más tiempo, comprobaciones para detectar las colas de mensajes están ocupadas. Si hay mensajes esperando a ser procesados, se leen en la cola y se envían a la ventana correspondiente. Una vez procesado el mensaje en la ventana destino, el bucle continuará con toda normalidad, al menos hasta que sea interrumpido de nuevo por la aparición de otro evento o hasta que finalice.

Para inspeccionar en la cola de mensajes se utiliza como antes la función *PeekMessage.* Si devuelve un valor cierto, es decir hay mensajes en la cola, se envía el mensaje leído a su ventana correspondiente utilizando SendMessage.

Esta solución funciona perfectamente en la mayoría de las situaciones, aunque debe de modificarse un poco en algunos casos. Cuando el bucle se utiliza para responder a un mensaje de petición para el pintado de la pantalla *(WM_PAINT)*, o cuando se realicen asignaciones de memoria dinámica dentro del bucle, es posible que aparezcan problemas. En el primer caso, es posible que tenga que abandonarse el bucle (por que ha llegado un mensaje), sin haber repintado toda la ventana. Esto puede tener graves consecuencias para el sistema, si no se tiene cuidado de llamar a *EndPaint()* antes de abandonar el bucle. La solución ideal sería llamar a EndPaint() para borrar el mensaje WM_PAINT de la cola de mensajes, y luego invalidar la zona que falta por pintar utilizando, por ejemplo, invalidateRect().

```
Inicio del bucle
        Reservar Memoria / Abrir ficheros
        Realizar el proceso
        Liberar memoria / Cerrar ficheros
        Si existen mensajes en la cola
        enviárselos a la ventana correspondiente
Fin del Bucle
```

Figura 4. Bucle concurrente con utilización de memoria dinámica y ficheros.

En caso de hacer asignaciones y liberaciones de memoria dinámica dentro del bucle, es recomendable liberar la memoria dinámica antes de salir del mismo mediante la función SendMessage(). Lo mismo en caso de trabajar con ficheros. No es recomendable mantener ficheros abiertos entre mensajes, ya que otra aplicación puede tomar el control del sistema.

El código para esta clase de bucles puede organizarse como se indica en el pseudocódigo de la figura 4.

LA PROGRAMACIÓN ADECUADA DEL "TIMER" DE WINDOWS TAMBIÉN PERMITE CREAR APLICACIONES CONCURRENTES

Además de los métodos que analizan en transito de los mensajes en el sistema, existe otra posibilidad para implementar concurrencia. Consiste en programar el reloj del sistema de Windows para que envíe un mensaje a la aplicación con una frecuencia constante, que será fijada por el programador.

La programación del reloj se escapa un poco del tema de este articulo, simplemente decir que permite, entre otras cosas, lograr que parte del código de la aplicación se ejecute repetidas veces y con la frecuencia deseada, Así, si antes la aplicación se activaba cuando no había eventos, ahora se activa N veces por segundo. Un ejemplo característico de esta técnica es la grabación automática de los archivos, cada X minutos, un procesador de texto.

Sin embargo, el reloj de Windows, aunque permite y a veces facilita la programación concurrente, se sale un poco del contenido del artículo, y se deja para otra ocasión. De todas formas, con lo visto hasta ahora, ya se conocen una serie de técnicas que ayudarán a los lectores a aumentar las prestaciones de sus aplicaciones.



En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

P *Ante el tremendo bombardeo en el mercado de Windows 95, ¿nos encontramos ante el final de los programas codificados para DOS? ¿tendremos que aprender a programar bajo Windows si queremos seguir "en activo"? Gracias de antemano.*

Alejandro Sanz
(Mallorca)

R Desde luego, si desea mantenerse "cotizado" en el mercado de los programadores, debe aprender a programar bajo Windows, aunque, debido a las herramientas de programación visual, no sea necesario realizarlo a bajo nivel. De todas formas, si lo suyo son las demos, los Vídeo-juegos o cualquier aplicación gráfica de alto nivel, debe seguir programándolas en DOS, pues las prestaciones, la dedicación completa del procesador y la libertad de acción es inimitable. En nuestra opinión, siempre deberá existir un entorno de ejecución "a pantalla completa", sin la dependencia (y limitaciones) que imponen las ventanas; en el cual se tenga un completo control sobre el hardware y el sistema operativo.

En otras plataformas de ordenadores, como el Unix o incluso en los mismísimos Silicon Graphics se carece de esta posibilidad y no proliferan mucho ni los juegos ni las demos, pese a que disponen de una potencia gráfica y musical que humillaría a cualquier Pentium...

P *Me gustaría iniciarme en el mundo de los hackers y creo que un buen comienzo sería el modificar el funcionamiento de algún juego. Había pensado por ejemplo en aumentar el número de "vidas" del personaje o cambiar el color del fondo de la pantalla. Necesito algún tipo de ayuda, por lo menos relativa a como empezar. Gracias y adelante con la revista.*

Antonio Ferrero
(Madrid)

R Es imposible explicar en esta sección algo tan complejo. Para modificar o "pokear" un programa del tipo que sea (sin disponer del código fuente, claro), se requiere un buen nivel de programación en ensamblador y del funcionamiento de las distintas llamadas al Sistema Operativo. También será muy útil el conocer como trabajan los compiladores, si es que el programa estaba desarrollado originalmente en C o en Pascal.

En cuanto a conocimientos, eso es todo (no está mal), pero además hacen falta algunas herramientas de software: Sería imprescindible un buen (el mejor siempre se quedará corto) debugger o depurador que permitirá ir trazando paso a paso la ejecución del programa, visualizando el código desensamblado y el estado que van adquiriendo las variables referenciadas.

También hace falta un editor de ficheros binarios, del tipo DiskEdit en el que se pueda buscar una cadena hexadecimal determinada y modificarla directamente, salvando después.

A grandes rasgos, el proceso sería el siguiente:

- Localizar la zona de código en donde se realiza la tarea a modificar y la dirección de memoria en donde se almacenan los valores que se quieren cambiar. Por ejemplo, si sabe la línea de código que decrementa la energía del personaje, ya está casi todo solucionado. O se anula la línea entera (mediante NOPs) o se accede en otro punto y se pone al máximo el contador.

- Mediante el editor de ficheros se busca la cadena que contiene el código a cambiar y se modifica con la nueva (habrá que calcular el valor hexadecimal de las instrucciones que se desean insertar). Algunos depuradores disponen de la posibilidad de teclear directamente líneas de ensamblador encima del fuente desensamblado original.

- No habrá que olvidarse de realizar una copia de seguridad del programa original antes de modificarlo de esta manera, pues habrá muchas posibilidades de que nunca más vuelva a funcionar.

Para introducirse un poco más en el mundo de los "inquietos" lo mejor es engancharse a algún área de mensajes de Fidonet dedicada al tema, o conectar con alguna BBS que disponga de un club de programadores, en donde seguro que habrá gente dispuesta a ayudar.

P *Les ruego me resuelvan un "problemilla" que me está volviendo loco. Resulta que me defino un array de ocho vectores de tres valores enteros cada uno y los relleno con valores constantes "a pelo". Hasta aquí todo correcto, pero al comprobar el contenido del array me encuentro (o se me informa) que los datos esperados son sólo eso, esperados, y que me aparecen valores que deberían estar en otro sitio. ¿Es un error del compilador?, ¿del gestor de memoria? o ¿es puramente mío?*

Antonio Morales Carmona
(Córdoba)

R Me temo que la causa del error es suyo, pero si le sirve de consuelo, es habitual en la gente que comienza a programar en C (y a algunos que llevan ya muchos años...) El fuente "maldito" es el siguiente:

```
#include <stdio.h>

void main(void)
{
  int a[2], b[2], c[2], d[2], e[2], f[2],
g[2], h[2];

  a[0] = 10;   a[1] = 20;   a[2] = 30;
  b[0] = 10;   b[1] = 20;   b[2] = 30;
  c[0] = 10;   c[1] = 20;   c[2] = 30;
  d[0] = 10;   d[1] = 20;   d[2] = 30;
  e[0] = 10;   e[1] = 20;   e[2] = 30;
       .....

printf ("\n b[0] = %3d  b[1] = %3d  b[2]
= %3d", b[0], b[1], b[2]);
}
```

Vemos que en primer lugar se definen arrays de 2 elementos (DOS), esto es: a[0] y a[1]. Por lo que a[2] no existe. El lenguaje C no comprueba los índices de los arrays o cadenas y sus compiladores se limitan a convertir cualquier subíndice de una cadena a una dirección de memoria, ya sea relativa o absoluta. Esto repercute en una compilación rápida y en una ejecución muy optimizada, además de permitir infinitas referencias y direccionamientos indirectos.

Cuando se accede a índices "inventados" los peligros son evidentes. Si las variables están definidas una detrás de otra, posiblemente se accede a las que están a continuación de la que está siendo desbordada, pero cuando llegamos al final de la zona de variables, los resultados son impredecibles.

En el caso que nos ocupa se obtendrá:

b[0] = 10 b[1] = 20 b[2] = 10

pues la dirección b[2] ha sido "pisada" por la c[0], que fue cargada posteriormente.

P *Estoy empezando con la programación gráfica y necesito calcular infinidad de direcciones físicas de pantalla, para poner un pixel o para conocer la dirección inicial de un determinado bloque de pantalla. El cálculo es muy sencillo en los modos gráficos habituales: (x * 320) + y, pero resulta lento y pierdo la mayoría del tiempo de ejecución en dicha operación. Incluso en ensamblador no me cuadran los tiempos. Si me dan alguna sugerencia, les estaré muy agradecido. Un saludo.*

Mariano Mendoza
(Teruel)

R Desde luego que resulta lento, sobre todo utilizando alguna operación de las denominadas "pesadas" en el mundo de la informática de gestión. Las multiplicaciones y las divisiones consumen muchísimos ciclos del procesador (cientos en algunos casos), frente a los pocos (1 o 2) que consumen las operaciones rápidas, tales como sumas, restas o rotaciones de bits.

Lo más sencillo para obtener el offset relativo de pantalla sería:

```
; Se coge la coordenada Y
mov     dx, Y
; Se multiplica por 320 (anchura de un
scan)
        mov     ax, 320
        mul     dx
        ; Se le suma la coordenada X,
para obtener el desplazamiento total
        add     ax, X
```

Pero lo más rápido (posiblemente) sería:

```
; Se coge la coordenada Y
mov     dx, Y
; Se la multiplica por 256, intercam-
biando
; sus bytes alto y bajo. Como en el alto
no
; debe haber nada (Y máxima = 200)
es igual a
; desplazarla hacia la izquierda 8
veces
; (cada desplazamiento es igual a mul-
tiplicarla
; por 2) 2 elevada a 8 = 256
xchg    dh, dl
; Se Coge el valor de Y * 256
mov     ax, dx
; Se desplaza hacia la derecha un par
de veces
; (división por 4), quedando una mul-
tiplicación
; por 32 (Y * 256 / 4 = Y * 32)
shr     ax, 1
shr     ax, 1
; Ahora se suman ambos valores y
obtenemos
; la multiplicación por 320:
; (Y * 256) + (Y * 32) = Y * 320
add     ax, dx
; Sólo resta sumarle el valor X
add     ax, X
```

Es curioso lo que se puede conseguir con un poco de imaginación y unas cuantas operaciones básicas, pero gracias a este tipo de trucos se optimizan y mejoran día a día los programas y sobre todo, los vídeo-juegos.

De cualquier manera, en futuros números de Sólo Programadores queremos comenzar una sección dedicada exclusivamente a la programación de vídeo-juegos, dado el especial interés mostrado por tantos lectores.

TOWERTEX

¡CONÉCTATE A TOWER!

Descubre las enormes posibilidades a las que puedes acceder a través de tu ordenador

TOWER ha creado para ti un soporte videotex. Para conectar con él sólo necesitas un modem, un software de comunicaciones, tu ordenador y una línea telefónica. En esta gran base de datos interactiva encontrarás:

CONEXIÓN CON LAS REVISTAS:
Noticias, sumarios, números atrasados, debates sobre temas de interés, seminarios, congresos, cartas al director y recomendaciones de los lectores. También existe un espacio para los diálogos y la mensajería. Además, podrás conectar con otros usuarios y enviarles cartas y mensajes.

MERCADO DE OCASIÓN
Para comprar y vender.

CLUB DE USUARIOS
Intercambio de información, actividades conjuntas, contacto con asociaciones, viajes, etc.

CONEXIÓN CON INTERNET
Podrás navegar por la autopista de la información más conocida: la "Red de Redes".

TELESOFTWARE
Con esta opción te ofrecemos un amplio menú de programas que podrás adquirir desde tu ordenador, sin salir de casa y sin tarjetas de crédito, ¡PAGANDO SÓLO EL COSTE DE LA CONEXIÓN!

JUEGOS Y CONCURSOS
Para disfrutar con tus amigos.

Contarás con el mejor software de comunicaciones, la máxima rapidez de envío de información y la mayor calidad en imágenes y gráficos. Para conectarte sólo necesitas un modem, un programa de conexión y utilizar el nemónimo *TOWER# por el nivel 032. En el próximo número de PC MEDIA encontrarás el programa de comunicaciones y un artículo para enseñarte a "navegar" por TOWERTEX.